Connected Urban Twins

HafenCity Universität Hamburg | City Science Lab
A Cooperation with the MIT Media Lab

Dokumentation

Prototyp Urban Model Platform & Masterportal Simulation Controller

Teilprojekt 4.4
Modellierung, Simulation, KI und Machine Learning

Versionsverlauf

Version	Beschreibung	Autor
1.0	Erste vollständige Version des Dokuments	Rico Herzog

Source Code

Der Quellcode für die vorliegende Dokumentation ist im „CUT RE3“ Repository des City Science Labs unter https://github.com/cityscienceclab/cut-re3 abrufbar.

Abkürzungsverzeichnis

ADES	Application, Deployment and Execution Service
API	Application Programming Interface
CWL	Common Workflow Language
EMS	Execution Management Service
GIS	Geographisches Informationssystem
JSON	JavaScript Object Notation
OGC	Open Geospatial Consortium
REST	REpresentational State Transfer
SimCo	Simulation Controller
UDP	Urban Data Platform
UMP	Urban Model Platform
WFS	Web Feature Service
WMS	Web Map Service
WPS	Web Processing Service
XML	eXtensible Markup Language

Abbildungsverzeichnis

Tabellenverzeichnis

Inhaltsverzeichnis

Executive Summary

Digitale Modelle und Simulationen sind ein Fokuspunkt in der Entwicklung von digitalen Stadtzwillingen. Sie ermöglichen „Was wäre wenn?“ Szenarien, schaffen Mehrwerte in der Datenanalyse und ermöglichen ein besseres Verständnis der vielfältigen und komplexen Zusammenhänge in der der Stadtentwicklung. Aktuell existiert in deren unterschiedlichsten Teilbereichen bereits eine Vielzahl an Simulationsmodellen. Beispiele sind Verkehrs- und Erreichbarkeitsmodelle, Prognosemodelle der Bevölkerungsentwicklung oder Wind- und Lärmsimulationen von geplanten Bauvorhaben.

Um diese Potentiale der einzelnen Modelle über die einzelnen Silos heraus nutzbar machen zu können und die Modelle miteinander verknüpfen zu können, braucht es neue technische und organisatorische Lösungen. Unter dem Ansatzpunkt einer offenen urbanen Plattform (*DIN SPEC 91357*, 2017) entwickelte das City Science Lab im Projekt „Connected Urban Twins“ einen Prototypen einer Urban Model Platform und eines mit der Plattform verbundenen Simulation Controllers.

Die Urban Model Platform basiert auf dem offenen Standard „API Processes“ des Open Geospatial Consortiums. Dieser legt fest, wie Prozesse und Modelle standardisiert über eine Webschnittstelle abgerufen werden können. Die Modellplattform implementiert den Standard und agiert gleichzeitig nur als Mittler: Sie ist mit beliebig vielen Modellservern verbunden und leitet die Anfragen der User zur Berechnung an den jeweiligen Server weiter. Gleichzeitig sammelt sie in einer Datenbank und daran angeschlossenen Geodiensten die Ergebnisse, welche damit unter einem „Single Point of Truth“ und mit den jeweiligen Metadaten abrufbar sind.

Der Simulation Controller wurde als Add-On zum Open-Source Web GIS Masterportal entwickelt. Dieses kann nun mit einzelnen Simulationslayern konfiguriert werden, die über das Add-On mit der Urban Model Plattform verknüpft sind. User bekommen dynamisch die konfigurierten Modelle angezeigt, können eigens gewählte Input-Parameter eingeben und beliebige Simulationsläufe starten. Sobald die Ergebnisse vorliegen, können diese dynamisch im Masterportal auf der Karte und als Diagramme visualisiert und gefiltert werden.

Die prototypischen Entwicklungen zeigen, welche Möglichkeiten durch die Implementierung einer Urban Model Platform im Rahmen digitaler Stadtzwillinge geschaffen werden: Modelle, die bislang nur in Silos verfügbar sind, können durch offene Schnittstellen auf einer Plattform zusammengeführt und miteinander kombiniert werden. Dadurch werden auf einer technischen Ebene vielschichtige „Was wäre Wenn?“ Szenarien und Multi-Modell Ansätze ermöglicht. Ein einzelnes Modell ist für sich immer ein unvollständiges Abbild der Realität. Viele Modelle ergänzen sich und ermöglichen so einen umfassenderen Umgang mit städtischer Komplexität und damit bessere Entscheidungsfindungen.

1 Einführung und Hintergrund

Dieses Dokument dient der umfänglichen Dokumentation und Aufbereitung der prototypischen Entwicklungen einer Urban Model Platform und eines damit verbundenen Simulation Controllers des City Science Labs der HafenCity Universität im Rahmen des „Connected Urban Twin" Projekts. Im Kontext der dort stattfindenden transformativen und experimentellen Forschung zu Modellen, Simulationen, KI und Machine Learning im Teilprojekt 4 wurde das standardisierte Zusammenspiel verschiedenster Modelle und deren Verknüpfung mit verschiedensten Tools als notwendige technologische Entwicklungen im Rahmen des dritten Realexperiments zum Thema Klimaschutz und soziale Gerechtigkeit identifiziert. Nach einer internen Konzeptionsphase und Konsultation der Teilprojekte 1 und 2 wurden die technologischen Entwicklungen in Zusammenarbeit mit dem IT-Dienstleister Ubilabs umgesetzt. Im Folgenden werden in einem kurzen Abriss ein theoretischer Hintergrund zu Modellen und Simulationen in digitalen Stadtzwillingen gegeben. Vereinzelt wird auf weiterführende Fachliteratur verwiesen, die jedoch keineswegs vorausgesetzt wird, sondern bei Bedarf lediglich als weiterführender Verweis dienen soll.

Digitale Modelle und Simulationen sind ein Fokuspunkt der Entwicklung von digitalen Stadtzwillingen. Mit der Hilfe von solchen Modellen können die vorhandenen städtischen Daten vielfältig genutzt und zur Grundlage von „Was wäre wenn?" Szenarien gemacht werden können. Dabei ist unter digitalen Modellen ganz grundsätzlich eine nützliche Abbildung eines Zielsystems mithilfe digitaler Technologie zu verstehen (in Anlehnung an Frigg & Hartmann, 2020). Es existieren unterschiedlichste Arten der Abbildung für unterschiedlichste Zwecke. Manche Modelle bilden ein sehr spezifisches Zielsystem ab und dienen damit der Beantwortung ganz konkreter Fragestellungen. So werden in den städtischen Verkehrsbehörden beispielsweise ausgereifte Modelle des Verhaltens von unterschiedlichsten Verkehrsteilnehmenden genutzt, um den allgemeinen Verkehrsfluss beispielsweise durch Änderungen an Ampelschaltungen zu optimieren. Andere Modelle wie die aktuellen Textgenerierungsmodelle wie ChatGPT können deutlich universeller eingesetzt werden und auf Grundlage eines individuellen Inputs eines Users Text zu den unterschiedlichsten Themen und Zwecken generieren. Gemein ist jedoch allen digitalen Modellen, dass sie auf vereinfachenden Annahmen beruhen und die Realität immer nur bis zu einem gewissen Grad abbilden. Der englische Statistiker George Box prägte in diesem Zusammenhang die Aussage „All models are wrong, but some are useful".

Aktuell werden viele solcher Modelle bereits in der Stadtentwicklungspraxis eingesetzt. Beispiele dafür sind unter anderem die Verkehrssteuerung, die Prognose von Bevölkerungsentwicklungen, Erreichbarkeitsmodelle oder die Lärm- und Windsimulation von geplanten Bauprojekten. Viele solcher Modelle sind über die Jahre in bestimmten Domänen gewachsen und werden dort isoliert von anderen Fachrichtungen eingesetzt. Damit ist die aktuelle Situation ähnlich der der städtischen Daten: Oftmals liegen diese nur in einem spezifischen Silo für ein bestimmtes Fachverfahren vor. Der

größte Mehrwert in der Anwendung ist jedoch in der Kombination und fachübergreifenden Nutzung zu finden. Was für städtische Daten das Verschneiden von beispielsweise Verkehrs- und Umweltdaten bedeutet, ist übertragen auf Modelle die Integration von Verkehrs- und Umweltmodellen. Ein digitales Experiment im Rahmen einer Simulation wie beispielsweise die Änderung der Ampelschaltung liefert dann nicht nur das veränderte Verkehrsverhalten, sondern auch die damit zu erwartenden Auswirkungen auf die Luftqualität.

Begreift man digitale Stadtzwillinge als Konzept zur Nutzbarmachung der digitalen Ressourcen einer Stadt (vgl. Schubbe et al., 2023), bedarf es im Kontext von digitalen Modellen und Simulationen eine technische und soziale Infrastruktur, um verschiedenste Modelle domänenübergreifend zu organisieren, ansprechbar zu machen und miteinander zu kombinieren. Um in der Analogie der städtischen Daten zu bleiben, scheint diesbezüglich der Ansatz eines „Systems von Systemen" zielführend zu sein, wie er in der *DIN SPEC 91357 (*2017) zu Offenen Urbanen Plattformen beschrieben wurde. Hierbei ist oberste Maxime, nicht die einzelnen Datenmangementsysteme der unterschiedlichen Domänen zu ändern, sondern ein übergeordnetes System zu schaffen, welches die verschiedenen Systeme auf einer Plattform über offene Schnittstellen verknüpft. Ziel dieses Systems ist es damit, die verschiedenen vertikalen Silos als horizontale Brücke zu verbinden.

Auf Grundlage dieser Vorüberlegungen wurde im Teilprojekt 4.4 ein Prototyp einer „Urban Model Platform" (UMP) als „System von Systemen" auf Basis offener Standards entwickelt. Als zweite große Komponente entstand mit dem „Simulation Controller" ein Add-On zum Open-Source Web-GIS Masterportal, welches das browserbasierte Frontend mit dem Prototyp der UMP verbindet.

2 Ziele

Ziel des Prototyps war es, einen ersten technischen Proof-of-Concept auf Basis existierender offener Standards zu entwickeln. Dieser soll folgende grundlegende Funktionalitäten abdecken:

- Bereitstellung und Simulation von unterschiedlichen Modellen über offene Schnittstellen
- Integration der unterschiedlichen Modelle auf einer offenen Plattform
- Technische Verbindung der Plattform mit dem Masterportal, um Modelle anzeigen, ausführen, und deren Ergebnisse darstellen zu können

Dabei wurde während der Entwicklung insbesondere auf die Wiedernutzung von bereits existierenden Technologien und spätere Skalierbarkeit geachtet. Fragen der organisatorischen Integration und Governance der unterschiedlichen Modelle in einem produktiven System wurden aufgrund des Fokus auf einen technischen Proof of Concept nicht betrachtet.

3 Übersicht der Software-Architektur

Wie in Abbildung 1 dargestellt, besteht der entwickelte Protoyp aus zwei unterschiedlichen Elementen, auf die in den folgenden Kapiteln genauer eingegangen wird.

Als **Backend-Komponente** wurde eine Urban Model Platform (UMP) entwickelt, welche per Config-Datei mit einer beliebigen Anzahl an Modellservern verknüpft werden kann. Die Schnittstelle zwischen den jeweiligen Modellservern und der UMP bildet jeweils die OGC API Processes. Die Ergebnisse, die von den einzelnen Modellen errechnet werden, werden innerhalb der UMP in einer PostGIS Datenbank gespeichert und über Geodienste eines Geoservers zur Verfügung gestellt.

Als **Frontend-Komponente** wurde basierend auf dem Masterportal ein Simulation Controller Add-On (SimCo) entwickelt, welches per Config-Datei mit ausgewählten Modellen der UMP verknüpft werden kann. In der Portalinstanz werden diese ausgewählten innerhalb des AddOns dynamisch angezeigt, können ausgewählt und nach Eingabe der erforderlichen Parameter ausgeführt werden. Sobald die Simulationsergebnisse vorliegen, werden diese in einer Liste aufgeführt und können individuell in der Portalinstanz visualisiert und gefiltert werden.

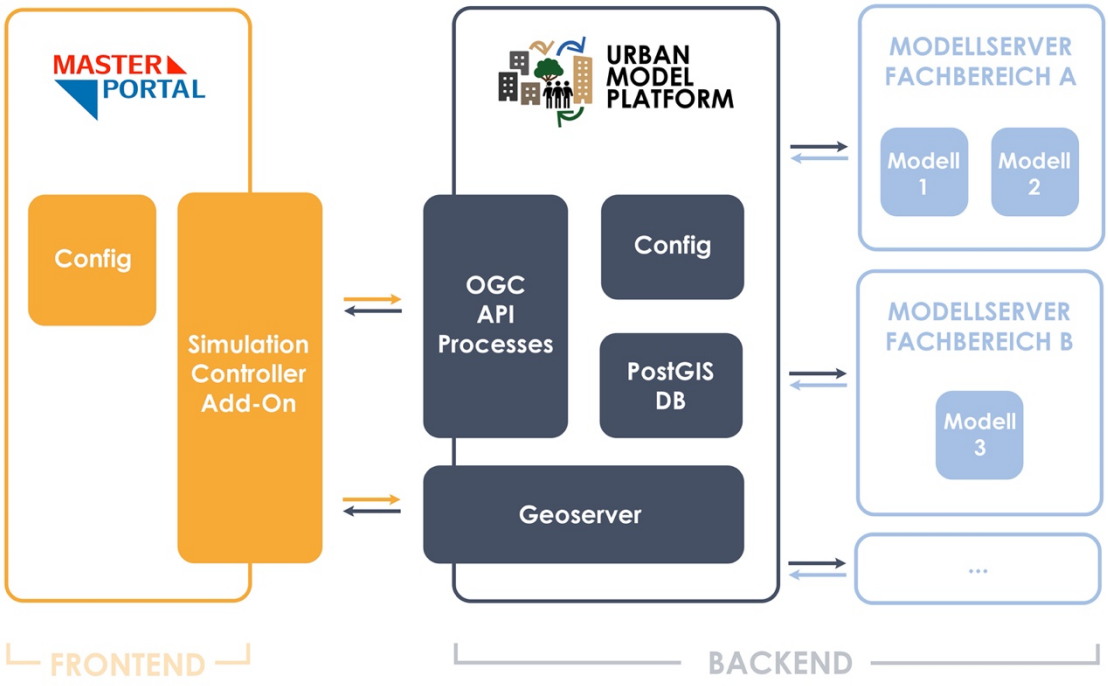

Abbildung 1: Architekturübersicht

4 Urban Model Platform

Die Urban Model Platform (UMP) bildet das Herzstück des Prototyps. Sie ist nach den leitenden Prinzipien einer offenen urbanen Plattform (*DIN SPEC 91357*, 2017) konstruiert und ermöglicht das Anbinden einer Vielzahl von (Fach)Modellservern. Im Wesentlichen besteht sie aus den vier Komponenten OGC API Processes, Config-Datei, Geoserver und einer PostGIS Datenbank. Im Folgenden wird zunächst auf die einzelnen Komponenten eingegangen, bevor die angebundenen Modellserver erläutert werden.

4.1 OGC API Processes

Das Open Geospatial Consortium (OGC) bietet seit 2005 und der Veröffentlichung des zum „Web Processing Service" (WPS) gehörenden Standards die Möglichkeit, Berechnungen und Modelle über eine Schnittstelle ausführen zu lassen. Hintergrund sind oftmals rechenaufwendige Operationen an raumbezogenen Daten, die durch einen WPS an einen Server ausgelagert und standardisiert abgerufen werden können. Als Weiterentwicklung des auf XML basierenden WPS veröffentlichte das OGC 2021 im Rahmen ihrer umfangreichen Umstellung auf moderne JSON und REST-basierte APIs den Core-Standard der API Processes (Open Geospatial Consortium, 2021).

Tabelle 1: Endpunkte eines OGC API Processes konformen Servers (Open Geospatial Consortium, 2021)

Resource	Path	HTTP Method	Parameter
Landing Page	`/`	GET	N/A
Conformance Classes	`/conformance`	GET	N/A
Process List	`/processes`	GET	N/A
Process Description	`/processes /{processID}`	GET	processID (in path)
Process Execution	`/processes /{processID} /execution`	POST	processID (in path), Execute request (contained in body)
Job Status Info	`/jobs/{jobID}`	GET	jobID (in path)
Job Results	`/jobs/{jobID} /results`	GET	jobID (in path)

Die in Tabelle 1 aufgeführten Endpunkte stellen den Kern einer jeden API Processes konformen Implementierung dar. Auf der Landing-Page („/“) wird sowohl auf die Conformance classes („/conformance“) als auch auf die verfügbaren Prozesse („/processes“) verwiesen. Über eine bestimmte Prozess-ID kann auf eine Prozessbeschreibung („/processes/{processID}“) zugegriffen werden und der Prozess kann ausgeführt werden („/processes/{processID}/execution“). Für jede Prozessausführung wird ein Job mit einer spezifischen ID erstellt, der dem Client bei erfolgreichem Start der Prozessausführung zurückgegeben wird. Sein Status kann über die Job-Status-Info („/jobs/{jobID}“) und - sobald er berechnet ist - seine Ergebnisse über den Job-Ergebnis-Endpunkt („/jobs/{jobID}/results“) abgerufen werden.

Aufgrund der standardisierten Schnittstelle ist es möglich, verschiedene OGC API Processes konforme Server miteinander zu verknüpfen und zu verketten. Einzelne Instanzen müssen den Prozess letztendlich nicht zwangsläufig selbst ausführen, sondern können die vom Client kommenden Anfragen an die entsprechenden Instanzen weiterleiten.

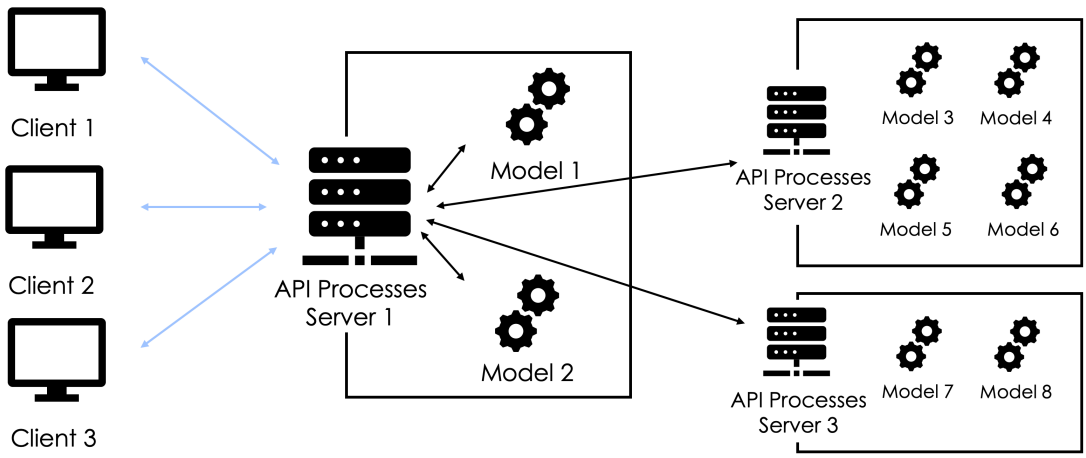

Abbildung 2: Verkettung von API Processes konformen Servern

Abbildung 2 verdeutlicht diese Architektur. Der mit den Clients verbundene Server 1 kann so den Zugang zu allen acht Modellen herstellen. Zwei der Modelle werden auf dem Server selbst ausgeführt. Die restlichen sechs Modelle verteilen sich auf API Processes Server 2 und 3. Sobald Server 1 jedoch die standardisierten Endpunkte von Server 2 und 3 bekannt sind, kann er selbst als Endpunkt für Modelle 3-8 fungieren. Anfragen der Clients werden dann an die entsprechenden anderen Server durchgeleitet und die Ergebnisse zurück an den Client gegeben.

Über diese Verkettung kann ein OGC API Processes Server als „System von Systemen“ konfiguriert werden, dessen Funktion das Zusammenführen von unterschiedlichsten Modellen auf unterschiedlichsten Servern im Sinne einer offenen urbanen Plattform ist.

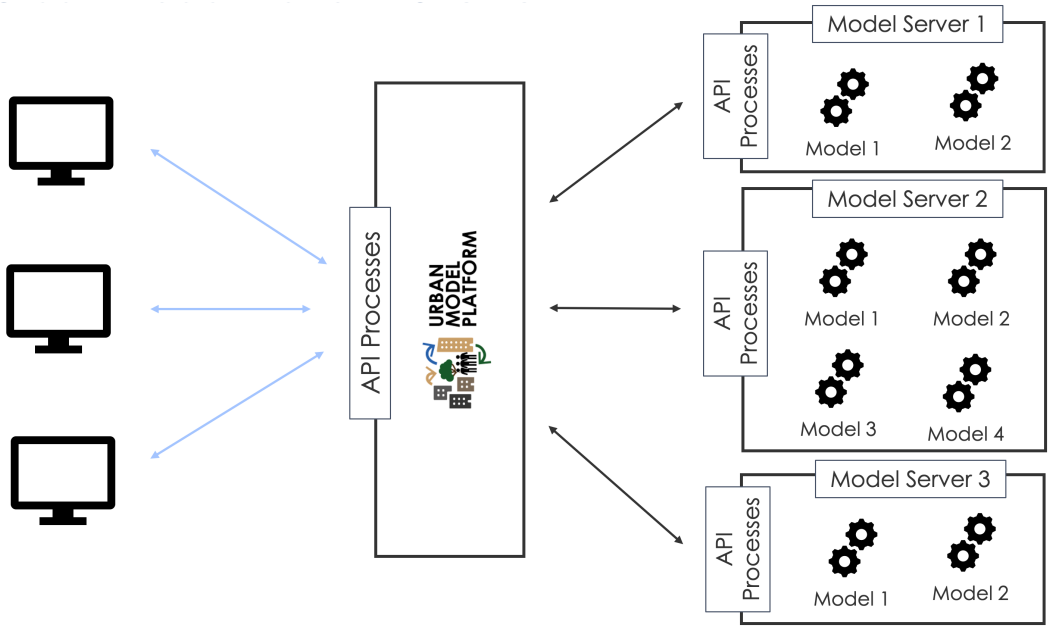

Abbildung 3: Ein OGC API Processes Server fungiert als Urban Model Platform

Abbildung 3 zeigt eine solche Funktionsweise eines speziell dafür entwickelten OGC API Processes Servers. Dieser Server kann damit sowohl als Plattform, als auch als Gatekeepter fungieren und bestimmten Clients Zugang zu einer Sammlung von auf unterschiedlichsten Servern verteilten Modellen verleihen. Gleichzeitig sind Modelle auch über die Schnittstellen den einzelnen Servern ansprechbar und können daher nicht ausschließlich über die Plattform abgerufen werden. Damit kann wie bisher auch jedes System weiterhin für sich stehen[1].

Im Rahmen der vorliegenden prototypischen UMP wurde die OGC API Processes in dieser Funktion eingesetzt. Die Implementierung erfolgte als Python Flask-Server in Verbindung mit einer PostGIS Datenbank. Damit können über diese Implementierung der OGC API Processes keine Modelle innerhalb der Instanz ausgeführt werden. Alle tatsächlich ausführbaren Modelle müssen auf anderen Modellservern liegen. Die UMP stellt damit konsolidiert die verschiedenen Prozesse von unterschiedlichen Providern zur Verfügung, leitet bei einer Client-Anfrage den „execution" Request an den jeweiligen Provider weiter und fragt in regelmäßigen Abständen automatisiert nach den Ergebnissen des Prozesses. Sobald ein die Ergebnisse vorliegen, werden diese in der Datenbank- und Geoserverkomponente der UMP unter Angabe der JobID und weiterer Metadaten gespeichert.

[1] Dies setzt natürlich voraus, dass die einzelnen Modellserver über eine OGC API Processes Schnittstelle ansprechbar sind. Es wird davon ausgegangen, dass dies in Realität aktuell nicht der Fall ist. Für den vorliegenden technischen Proof of Concept der UMP wurde diese Annahme getroffen. Im praktischen Betrieb gälte es in Zukunft, die einzelnen Modelle über Schnittstellen/Konnektoren mit der UMP zu verbinden. Hier bietet sich natürlich eine erneute Implementierung der OGC API Processes an.

Der Quellcode des gesamten Prototyps ist unter https://github.com/ci-tysciencelab/cut-re3 abrufbar. Die weiteren Hauptbestandteile sind in den folgenden Unterkapiteln beschrieben.

4.2 Config

Die Konfiguration der prototypischen UMP erfolgt in der „providers.yml" Datei. Wie in Abbildung 4 dargestellt ermöglicht die YAML-Struktur ein Anbinden beliebig vieler Modellserver. Diese werden jeweils mit einem einmaligen Identifier („modellserver-1" und „modellserver-2") versehen und mit verschiedenen Attributen ausgestattet.

```yaml
# This is the configuration file for setting up simulation servers.
# The servers should provide an OGC processes api which will be retrieved
# on the fly to provide all existing processes via this api. Please provide the base url
# of the api, e.g. if there is an endpoint https://example.org/rest/api/processes
# then provide https://example.org/rest/api as url.
# This file should be considered as a secret.

modelserver-1:
  url: "https://modelserver1.cut.hcu-hamburg.de"
  user: "cut"
  password: "password"
  timeout:  1800
  exclude:
    - "abm-test-model"

modelserver-2:
  url: "https://modelserver2.cut.hcu-hamburg.de"
  user: "cut"
  password: "password"
  timeout:  1800
```

Abbildung 4: Konfiguration der UMP

Notwendige Attribute sind:

- **url**. Hierüber wird der Link zur Landing Page eines OGC API Process konformen Servers hergestellt.
- **timeout**. Über dieses Attribut wird festgelegt, nach wie vielen Sekunden die automatisierten Ergebnisabfragen an die jeweiligen Modellserver abgebrochen werden sollen. Ein Timeout von 1800 bedeutet beispielsweise, dass sobald nach 1800 Sekunden (=30 Minuten) kein Ergebnis bei dem jeweiligen Endpunkt abrufbar ist, der Job als nicht erfolgreich gelabelt wird und weitere Abfragen unterbleiben.

Optionale Attribute sind:

- **user/password**. Diese Attibutskombination dient dazu, mit Basic Auth geschützte Landing Pages der Modellserver anzubinden.
- **exclude**. Hierüber können spezifische Modelle aufgelistet werden, die *nicht* über die UMP zur Verfügung gestellt werden sollen, obwohl sie als Endpunkt unter der jeweiligen URL zur Verfügung stehen. Der Name muss der entsprechenden processID des Endpunkts entsprechen.

4.3 Geoserver

Eine Geoserver-Instanz wurde als Teil der UMP integriert, um die verschiedensten Ergebnisse der Modellserver zentral zu sammeln und über verschiedene Dienste zur Verfügung zu stellen. Dies bedeutet in großen Teilen eine Doppelstruktur, da die Ergebnisse sowohl auf Seiten der Modellserver unter dem „/jobs/{jobID}/results“ Endpunkt, als auch im Geoserver der UMP vorgehalten werden. Trotz der Redundanz wurde diese Lösung präferiert, da durch die zentrale Nutzung des Geoservers eine serverseitige Filterung der Daten möglich wird und die Integration in verschiedenste bestehende Anwendungen über Standards wie WMS oder WFS vereinfacht wird. Zudem braucht es nur auf Seiten der UMS Pipelines, um die jeweiligen Ergebnis-Dateiformate auf den Modellservern in den Geoserver zu überführen. Gäbe es keine zentrale Sammelstelle der Simulationsergebnisse in Form eines Geoservers, müsste für jede einzelne Modellserver-Anwendungs-Kopplung sichergestellt werden, dass die Ergebnisdaten im richtigen Format über den richtigen Dienst bereitgestellt werden.

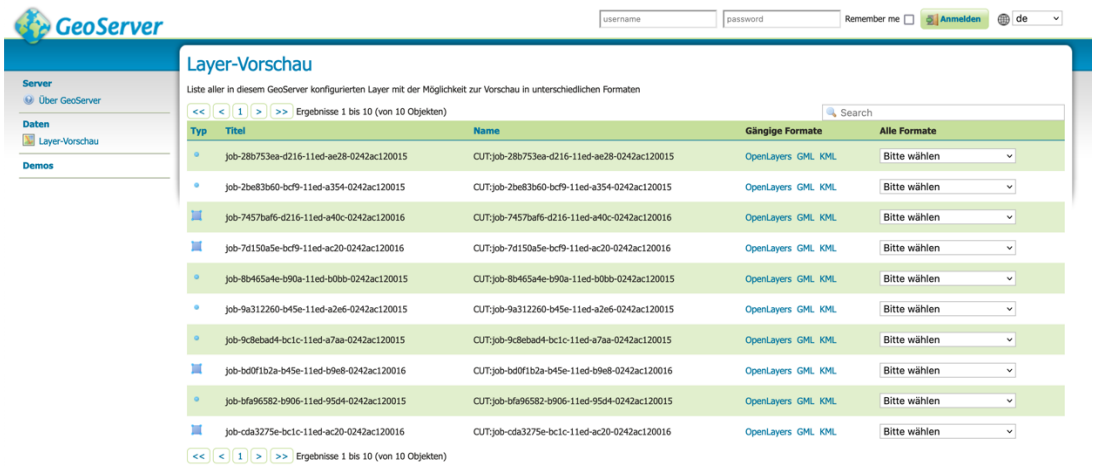

Abbildung 5: Layer im Geoserver

4.4 PostGIS Datenbank

Bei einer erfolgreichen Anfrage zum Start eines Prozesses auf einem Modellserver wird in der UMP in einer PostGIS Datenbank die jeweilige JobID, der Status, der Provider, Timestamps, die übergebenen Parameter und andere Metadaten gespeichert. Bei erfolgreicher Ausführung eines Prozesses werden die Ergebnisse (zur Zeit als GeoJSON) von der UMP unter dem jeweiligen Endpunkt abgefragt und ebenfalls der PostGIS Datenbank unter der jeweiligen JobID gespeichert.[2] Um eine clientseitige Filterung zu ermöglichen, werden Metadaten der Ergebnisse berechnet (zur Zeit sind das für jedes String-Attribut einmalige Werte und für jedes Integer/Float Attribut min und

[2] Ein weiterer Vorteil dieser Lösung ist die zentrale Integration von unterschiedlichsten Dateiformaten. Aktuell besteht eine Pipeline zum Einfügen von GeoJSON Ergebnissen; ähnliche Pipelines können auch zur Integration bspw. von Shapefile-Ergebnisdateien hinzugefügt werden.

max-Werte). Diese Metadaten werden ebenfalls in der PostGIS Datenbank unter der jeweiligen JobID abgelegt.

Sobald die Speicherung der Daten in der PostGIS Datenbank stattgefunden hat, werden die Ergebnisse über die Geoserver-Schnittstelle in einem vordefinierten Workspace als Layer abgelegt. Von dort aus stehen die Ergebnisse unter den konfigurierten Diensten zur Verfügung.

4.5 Modellserver

Als Beispiel-Modellserver wurden zwei Instanzen der **pygeoapi**[3] aufgesetzt, die je ein bzw. zwei Modelle über die OGC API Processes zur Verfügung stellen. Die Bibliothek ist eine Python-basierte Implementierung verschiedener OGC API Standards und kann unter der MIT Lizenz verwendet werden. Sie bietet die Möglichkeit, verschiedene Prozesse in Python als Teil des Flask Servers zu implementieren und über die API Processes bereitzustellen. Darüber hinaus integriert die pygeoapi für das Job-Management der verschiedenen Prozesse eine SQLite Datenbank. In dieser werden bei Ausführung eines Prozesses automatisch die Ergebnisse und die Metadaten zu den jeweiligen Jobs abgelegt.

Während des Aufsetzens der einzelnen Modellserver wurde unter anderem auch die Python-Implementierung **Weaver**[4] genauer untersucht. Diese versteht sich als serverseitige Lösung für das Management und das Ausführen unterschiedlichster Prozesse, die auf der OGC API Processes aufbaut. Einzelne Weaver-Instanzen können entweder als „Execution Management Service“ (EMS) oder als „Application, Deployment and Execution Service“ (ADES) konfiguriert werden. Erstere Konfiguration bedient im Wesentlichen die Funktionsweise der zuvor beschriebenen Modellplattform und leitet die Process execution requests von Clients an die entsprechenden ADES weiter. Diese führen dann die auf Grundlage der „Common Workflow Language“[5] spezifizierten Prozesse aus. Trotz des beträchtlichen Funktionsumfangs von Weaver wurde auf eine Umsetzung des Prototyps auf Basis dieser Software verzichtet. Grund dafür ist eine trotz eingehender Prüfung und Debuggings nicht funktionale Docker- oder Pythonimplementierung. Die Konzepte hinter Weaver scheinen dennoch ausgereifte Lösungen für grundsätzliche Problemstellungen zu sein, auf die bei einer weiteren Entwicklung der UMP oder einzelner Modellserver zurückgegriffen werden kann (s. auch Kapitel 6 - Key Learnings).

[3] Mehr Informationen unter https://pygeoapi.io/
[4] Mehr Informationen unter https://pavics-weaver.readthedocs.io/en/latest/index.html
[5] Mehr Informationen unter https://www.commonwl.org/

5 Simulation Controller

Der Simulation Controller (SimCo) ist ein Masterportal AddOn, welches die über die UMP bereitgestellten Modelle und Simulationen im Rahmen einer Masterportal-Instanz nutzbar machen soll. Dies erfolgt dynamisch und verleiht damit dem bislang rein Frontend-basierten Masterportal Zugang zu einem Backend in Form der UMP. Das AddOn und seine Funktionalitäten werden in 5.2 erläutert. Die Konfiguration einer Portalinstanz mit SimCo wurde so nah wie möglich an den bisherigen Layerkonfigurationen gehalten und ist unter 5.1 eingehender beschrieben.

5.1 Config

Analog zur Urban Data Platform (UDP) stellt die UMP möglicherweise zahlreiche Modelle zur Verfügung, die in ihrer Gesamtheit nicht in jeder Anwendung benötigt werden. Das Masterportal bietet daher die Möglichkeit, einzelne Portalinstanzen so zu konfigurieren, dass nur ein Teil der Daten der UDP angezeigt wird.[6] So entstanden in der Vergangenheit unterschiedliche Fachportale wie bspw. das Hamburger Schulportal, welche nur thematisch kuratierte Daten enthalten.

```
39   {
40     "id": "123456",
41     "isSimulationLayer": true,
42     "filterOnClient": false,
43     "simModelId": "cut:abm-test-model",
44     "styleId": "654321",
45     "name": "Simulation Layer",
46     "url": "http://localhost:3000/geoserver/CUT/ows",
47     "featureType": "",
48     "version": "1.0.0",
49     "typ": "WebGL",
50     "isVisibleInMap": true,
51     "legend": false,
52     "style": {
53       "symbol": {
54         "symbolType": "circle",
55         "size": 10,
56         "color": [0, 0, 255],
57         "opacity": 1,
58         "rotateWithView": true
59       }
60     }
61   }
62 ]
```

Abbildung 6: Konfiguration eines Modells im Masterportal

Um eine ähnliche Konfigurationsmöglichkeit für die einzubindenden Modelle und Simulationen herzustellen, können diese in der Konfiguration einer Portalinstanz analog angelegt werden. Damit wird ein jedes Modell zunächst wie in Abbildung 6 gezeigt als weiterer Datenlayer konfiguriert, der im Layertree angezeigt wird. Um einen entsprechenden Layer als Simulationslayer zu konfigurieren, sind drei weitere Attribute nötig:

[6] Für eine detaillierte technische Dokumentation des Masterportal wird auf https://bitbucket.org/geowerkstatt-hamburg/masterportal/src/dev/ verwiesen

- **isSimulationLayer**. Wenn dieser Wert auf „true“ gesetzt wird, versucht das SimCo AddOn ein entsprechendes Modell von der UMP zu laden
- **simModelId**. Hier wird ein String mit der entsprechenden ProcessID auf der angebundenen OGC API Processes Instanz erwartet. Wenn die UMP als API Processes Instanz verknüpft ist, wird ein Input vom Typ „provider:processId“ erwartet.
- **filterOnClient**. Wenn dieser Wert auf „true“ gesetzt ist, werden sämtliche Ergebnisdaten in den Client des Browsers geladen. Andernfalls werden die Daten serverseitig gefiltert und nur eine entsprechende Teilmenge vom Server zurückgegeben. Bei großen Datenmengen ist „filterOnClient“: false vorzuziehen. Bei kleineren zu erwartenden Datenmengen ist das Filtern der Daten im Browser des Clients deutlich performanter und führt zu flüssigeren Darstellungen.

Des Weiteren sind andere übliche Konfigurationen leicht modifiziert.

- **url**. Im Gegensatz zu „normalen“ Datenlayern wird hier keine eindeutige URL zu einem Geodatendienst eingetragen, sondern der Link zum Workspace des GeoServers, in dem alle Ergebnisse vorgehalten werden. Über die Modell- und JobID werden dann die jeweiligen Ergebnisse serverseitig gefiltert.
- **typ**. Hier ist prinzipiell kein Unterschied zu „normalen“ Datenlayern vorhanden. Insbesondere bei agentenbasierten Modellen, die eine Vielzahl an Agenten über mehrere Zeitschritte zurückgeben, empfielt es sich jedoch, eine WebGL Layerkonfiguration vorzunehmen. Da dieses Feature im Masterportal bislang nur experimentell umgesetzt wurde, ist hier jedoch ein anderes Styling zu verwenden.

In der „config.js“ Datei der Portalinstanz muss SimCo als AddOn wie gewöhnlich hinzugefügt werden. Ebenfalls muss die folgende Einstellung gesetzt werden, um das AddOn mit der UMP zu verbinden:

- **simulationApiUrl**. Hier wird der Link zur Landingpage der OGC API Processes konformen UMP festgelegt.

5.2 Masterportal Add-On

Das SimCo selbst wurde mit Vue.js entwickelt und baut auf der zuvor beschriebenen Layer- und Portalkonfiguration auf. SimCo verfügt über drei spezifische Ansichten, die im Folgenden erläutert werden und zwischen denen ein User jederzeit – sofern Modelle und Ergebnisse vorliegen – vor- und zurücknavigieren kann.

Übersichtsseite

Beim Öffnen des AddOns werden die in der Layerkonfiguration des Masterportals festgelegten Modelle dynamisch angezeigt. Das AddOn überprüft, ob unter der konfigurierten UMP-URL ein Modell mit der entsprechenden ProcessID vorhanden ist. Falls das der Fall ist, werden die Metadaten abgefragt und Name, sowie Beschreibung und weiterführende Informationen in einer Tabelle aufgelistet. Somit kann über einen zentralen Zugangspunkt auf alle zuvor konfigurierten Modelle zugegriffen werden. Verändert sich ein Modell im Lauf der Zeit (bspw. durch das Aktualisieren der Beschreibung), werden über die ProcessID immer die aktuellsten Informationen dynamisch in das Frontend geladen. Um eine Simulation ausführen zu können, muss das Modell mit dem Klick auf „Modell“ ausgewählt werden.

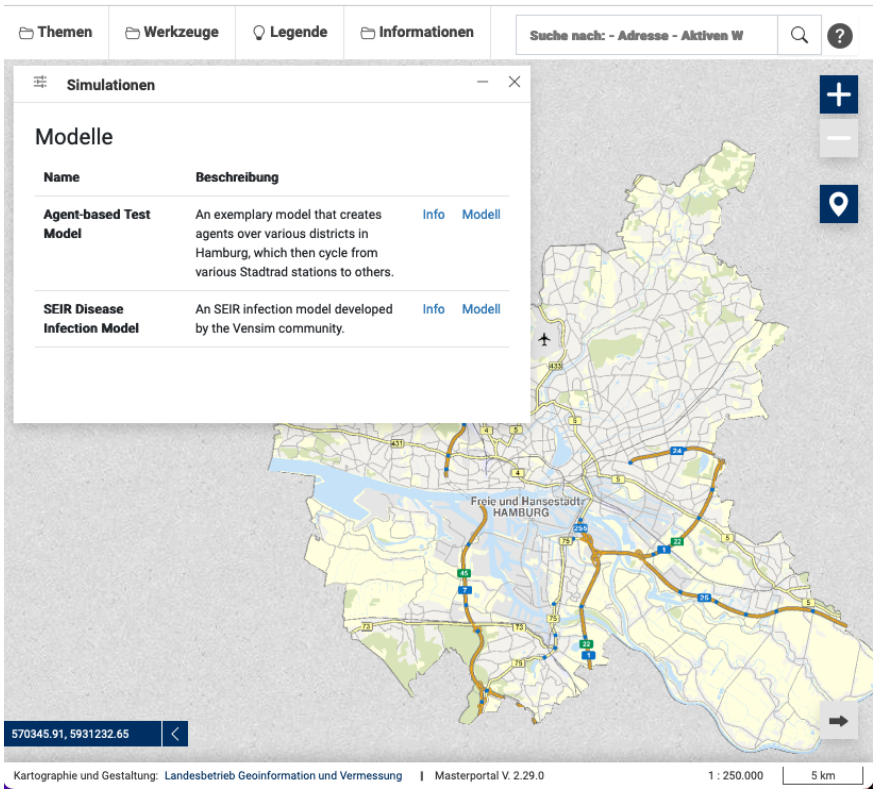

Abbildung 7: SimCo Übersichtsseite

Detailseite Modell

Sobald ein Modell ausgewählt wurde, gelangt der User zur Detailseite. Hier wird die Beschreibung des Modells, eine Liste an bisherigen Jobs und ein Bereich zum Starten einen neuen Jobs angezeigt.

Die Jobliste zeigt die bislang ausgeführten Jobs an. Über die UMP werden der jeweils aktuelle Status, ein Zeitstempel und die Input-Parameter angezeigt. Sofern der Status eines Jobs „successful“ anzeigt, kann auch das Ergebnis mit einem Klick auf „anzeigen“ abgerufen werden.

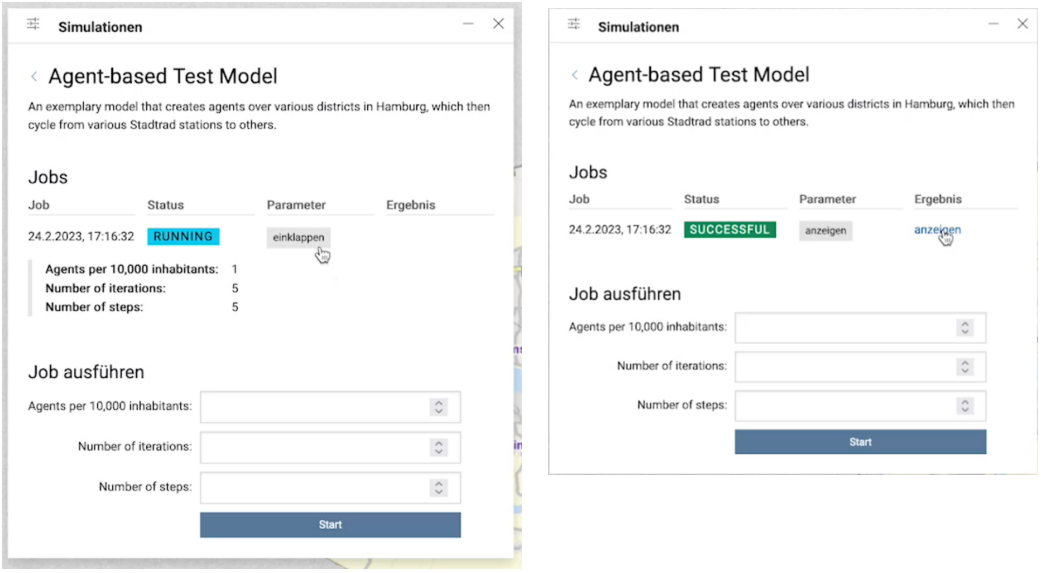

Abbildung 8: SimCo Detailseite Modell

Der Bereich „Job ausführen“ dient dem Starten eines neuen Jobs. Die Input-Parameter werden mit jeweiligen Input-Möglichkeiten auf Seiten des Users versehen und, sofern im Rahmen der API Processes Prozessdefinition bereitgestellt, direkt im Frontend validiert. Sofern ein Parameter erforderlich ist, kann der Prozess nicht gestartet werden, ohne dass ein entsprechender Wert eingetragen ist. Auch falsche Werte werden erkannt und mit entsprechenden Hinweisen versehen.

Detailseite Job

Sobald die Ergebnisse vorliegen und ein bestimmter Job ausgewählt wurde, werden auf der Detailseite Job die Ergebnisse des jeweiligen Jobs visualisiert. Hier hat der SimCo zwei Funktionalitäten: Zum einen zeigt er – wo immer möglich – die georeferenzierten Ergebnisse auf der Karte und – in prototypischer Form – als Diagramm an. Zum anderen besteht für den User die Möglichkeit, verschiedenen Attribute zu filtern und damit die Ergebnisse der Simulation genauer unter die Lupe zu nehmen.

Auf der Karte besteht je nach Konfiguration des Layers auch die Möglichkeit, einzelne Elemente über eine „GetFeatureInfo“ Box genauer zu inspizieren. Die Graphen sind aktuell nur für ein spezifisches Modell einsetzbar und „hardgecodet“. Da insbesondere in der graphischen Aggregation der Ergebnisdaten ein großer Mehrwert für den User liegt, gilt es hier als nächsten Entwicklungsschritt einen möglichst generischen Weg zu finden, um bestimmte Attribute je nach Modell individuell plotten zu können (s. auch Kapitel 7 für nächste Schritte).

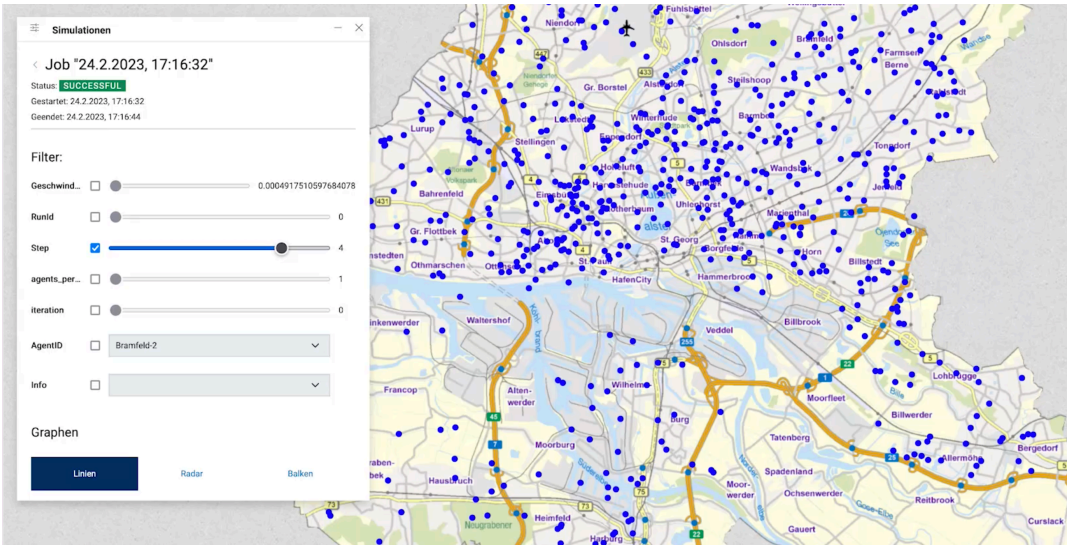

Abbildung 9: SimCo Detailseite Job

6 Key Learnings

Im Rahmen der Softwareentwicklung und des damit verbundenen Lösungsfindungsprozesses wurden Erfahrungen gesammelt, von denen die vier wichtigsten in diesem Kapitel gebündelt zusammengefasst sind. Sie sollen zusätzlich zur Beschreibung des fertigen Prototyps Einblicke in die Gründe für die spezifische Umsetzungsentscheidungen bieten.

L1: Die einzelnen Modellserver benötigen bestenfalls auch eine OGC API Processes Schnittstelle

Ausgehend von der Wahl der OGC API Processes als standardisierte Schnittstelle für die Urban Model Platform war zunächst unklar, wie die Modelle an die Plattform angebunden werden können. Die zentrale Fragestellung hierbei ist, wie der Informationsfluss zwischen dem Modell und der Plattform bestmöglich ausgestaltet werden kann. Zunächst bestand die Idee, die einzelnen Modelle mit einer Websocket-Verbindung individuell anzubinden, um einen beidseitigen Datenfluss zu ermöglichen.
Die Entscheidung, die Kommunikation über HTTP(S) und nicht über Websockets zu realisieren, wurde aufgrund möglicher Verluste und mangelnder Vorteile einer Sockets-Verbindung getroffen (s. auch L2). Dies führte zu der Feststellung, dass damit jeder einzelne Server einen Endpunkt und ein Jobmanagement benötigt, um den State-Transfer zwischen der UMP und den Modellen verlustfrei zu ermöglichen. Da diese Funktionalitäten bereits standardisiert über die OGC API Processes bereitgestellt werden können, ist eine Anbindung der Modellserver bestenfalls über die OGC API Processes abzubilden.

L2: Echtzeit-Streams sind nicht zwingend notwendig

Wie in L1 beschrieben, wurde sich im Prozess gegen eine Anbindung der Modelle über eine Websockets-Verbindung entschieden. Hauptgrund dafür war, dass bei einer Unterbrechung der Sockets-Verbindung zwischen UMP und dem Modell große

Datenverluste möglich sind. Wenn aufgrund eines kurzzeitigen Netzwerkausfalls die Sockets-Verbindung abbricht, werden die z.T. schon stunden- oder tagelangen Simulationsläufe hinfällig, sofern auf deren Seite keine eindeutige ID vergeben wird, unter der die Ergebnisse abgespeichert werden können. Wenn nun jedoch die ID vergeben wird, ist eine Implementierung eines Job-Managements mit eindeutiger Speicherung der IDs auf UMP und Modellseite notwendig (s. L1). Gleichzeitig bringt der Echtzeitstream für Simulationsmodelle wenig Vorteile: Eine Statusabfrage ist auch per HTTP(S) möglich. Zudem sind häufig die aggregierten Daten aus mehreren Simulationsläufen notwendig, um Aussagen treffen zu können. Ein Echtzeit-Stream bringt bei dem Großteil an (Simulations-)modellen wenig neue Erkenntnisse.[7] Sollte ein solcher Echtzeit-Stream in Zukunft als Anwendungsfall benötigt werden, könnte die Verwendung eines SensorThings Frost-Servers als weiteres Modul für eine UMP untersucht werden.

L3: Die Modellbereitstellung braucht weitere standardisierte Lösungen

Im vorliegenden Prototypen werden die Modelle auf den Modellservern mittels einer pygeoapi zur Verfügung gestellt. In diese wurden zwei Beispielmodelle „händisch" integriert, d.h. die einzelnen Python-Bibliotheken für die bestimmten System-Dynamics oder Agent-based Modelle wurden per pip requirements installiert, innerhalb der pygeoapi importiert und damit in die „Process"-Klasse integriert. Für eine Vielzahl an Modellen auf einem Modellserver erscheint dieser Weg aus mehreren Gründen nicht ideal:

1. Es muss immer die komplette pygeoapi-Instanz angepasst werden, sobald ein neues Modell zur Verfügung gestellt werden soll
2. Die verschiedenen benötigten Bibliotheken für unterschiedliche Modelle können sich gegenseitig ausschließen und es kann zu Versionskonflikten kommen
3. Nicht alle Modelle sind in Python ausführbar, d.h. es braucht einen Weg, um auch andere Scripte wie bspw. npm-JavaScript Packages oder komplexere Softwarepakete zu integrieren

Einige existierende Lösungen wie die unter Kap. 4.5 beschriebenen OGC API Processes Implementierung „Weaver" greift auf die Common Workflow Language (CWL) zurück, um eine standardisierte Ausführung von Arbeitsprozessen zu ermöglichen. Die CWL (https://www.commonwl.org/) ist ein offener Standard, um Workflow-Prozesse als .yml Datei zu beschreiben und mit kompatiblen Tools auszuführen. Da Docker in die CWL Prozesse integriert werden kann, wären damit auch komplexere organisierte und standardisierte Modellbereitstellungen möglich.

[7] Diese Aussage bezieht sich auf die Verwendung eines bereits validierten und verifizierten (Simulations-)Modells. Im Gegensatz ist in der Modellentwicklung ein Echtzeit-Anzeige von Simulationsläufen von großer Wichtigkeit, um das Modell verifizieren und validieren zu können. Da jedoch nur das bereits verifizierte und validierte Modell auf der Plattform zur Verfügung gestellt wird, ist die Echtzeit-Anzeige von wenig Relevanz.

L4: Die Bandbreite der möglichen Modelle ist riesig
Während der Entwicklungsphase des Prototyps wurde klar, dass die über die OGC API Processes bereitgestellte standardisierte Schnittstelle große Potentiale für eine sehr große Bandbreite an Modellen und Simulationen birgt. Da „nur" die entsprechenden Input- und Outputwerte über die API zur Verfügung gestellt werden, können sowohl spezifische Simulationsmodelle wie Wind, Lärm oder Luftqualität, als auch generalisierte Modelle wie generative Text-, Bild- oder Videomodelle angebunden werden. Eine Urban Model Platform kann so bspw. sowohl Modelle von Forschungsinstitutionen anbinden, als auch fachspezifische Modelle, die auf den Servern von privaten Unternehmen liegen – vorausgesetzt, diese werden über eine entsprechende Schnittstelle zur Verfügung gestellt. Die Fragen, die sich anschließen, berühren dann Finanzierungs-, Ausschreibungs- und Abrechnungsmodalitäten. Vorteil dieser Art der Bereitstellung ist, dass die Modelle auch interoperabel gestaltet werden können, d.h. ein privatwirtschaftliches Modell leicht mit einem städtischen kombiniert werden kann.

7 Nächste Schritte und Zusammenfassung

Aus den Entwicklungen und den Key Learnings in Kapitel 6 ergeben sich diverse Möglichkeiten, wie die weitere Entwicklung der Urban Model Platform und des Simulation Controllers ausgestaltet werden kann. Diese sind im Folgenden kurz skizziert.

- **Ausprobieren von standardisierten Lösungen der Modellbereitstellung und Versionierung**: Die Common Workflow Language als offenen Standard nutzen, um komplexere Modelle abzubilden und einen technischen Proof-of-Concept für die standardisierte Modellbereitstellung herzustellen. Über eine Versionierung der Modelle und einen Abgleich mit vergangenen Berechnungen können doppelte Simulationsläufe vermieden werden.

- **Weitere Input- und Outputformate integrieren:** Derzeit werden nur Strings, Arrays und Integers/Floats als Input im Simulation Controller unterstützt. GeoJSON ist bislang das einzige Output-Format, dass an die UMP angebunden werden und über Dienste zur Verfügung gestellt werden kann. Weitere Inputformate, beispielsweise Boundary-Boxes als Input oder 3D-Formate als Output können den Funktionsumfang beträchtlich erweitern.

- **Personalisierung**: Durch eine eindeutige Zuordnung von Simulationsläufen zu bestimmten Usern (entweder Personen oder Organisationen) können nur die Simulationsläufe angezeigt werden, die auch von dem entsprechenden User gestatet wurden. Hierzu ist ein Rollenkonzept für das Masterportal von großem Vorteil.

- **Verbesserte Darstellung der Ergebnisse im Simulation Controller:** Über ein dynamisches Styling, das direkt von der UMP über die errechneten Metadaten zur Verfügung gestellt wird, können die Ergebnisse in den Masterportal-Simulationslayern entsprechend graphisch aufbereitet werden. Zusätzlich gilt es, eine bessere Darstellung der Modellergebnisse in Form von verschiedenen Graphentypen zu ermöglichen, sodass die raumbezogene Darstellung nur ein Teil der Datenaufbereitung ist. Darüber hinaus können Möglichkeiten geschaffen werden, verschiedene Simulationsläufe miteinander zu vergleichen.

Zusammenfassend beweist der Prototyp Urban Model Platform und Masterportal Simulation Controller aus technischer Sicht, wie Modelle und Simulationen innerhalb des Konzepts eines digitalen Stadtzwillings für „Was wäre wenn?“ Szenarien verbunden und verknüpft werden können. Über eine Urban Model Platform als offene urbane Plattform können beliebig viele dezentrale Modelle, Algorithmen und Prozesse angebunden, über eine Plattform zur Verfügung gestellt und ausgeführt werden. Der Simulation Controller als Add-On zum Masterportal zeigt auf, wie die Urban Model Plattform im Rahmen der bereits bestehenden Infrastruktur nutzbar gemacht werden kann und einer Vielzahl an Usern Mehrwerte bieten kann.

Literaturverzeichnis

DIN SPEC 91357:2017-12, Referenzarchitekturmodell Offene Urbane Plattform_(OUP); Text Englisch. (2017). Beuth Verlag GmbH. https://doi.org/10.31030/2780217

Frigg, R., & Hartmann, S. (2020). Models in Science. In E. N. Zalta (Hrsg.), *The Stanford Encyclopedia of Philosophy* (Spring 2020). Metaphysics Research Lab, Stanford University. https://plato.stanford.edu/archives/spr2020/entries/models-science/

Open Geospatial Consortium. (2021). *OGC API - Processes—Part 1: Core*. https://docs.ogc.org/is/18-062r2/18-062r2.html

Schubbe, N., Bodecker, M., & Moshrefzadeh, M. (2023). Urbane Digitale Zwillinge als Baukastensystem: Ein Konzept aus dem Projekt Connected Urban Twins (CUT). *zfv – Zeitschrift für Geodäsie, Geoinformation und Landmanagement*, *1/2023*, 14–23. https://doi.org/10.12902/zfv-0417-2022