Requirements for Web-based 4D Visualisation of Integrated 3D City Models and Sensor Data in Urban Digital Twins

Joseph Gitahi[1] and Thomas H. Kolbe[1]

[1] Chair of Geoinformatics, Technical University of Munich, 80333 Munich, Germany
(joseph.gitahi, thomas.kolbe)@tum.de

Abstract. Urban Digital Twins (UDTs) have emerged as essential tools for managing city operations, forming the basis of smart city solutions. They offer a digital representation of the physical urban environment, which supports various city applications such as monitoring mobility, air quality, and modelling simulations. To accurately represent the physical world, UDTs need to be updated continuously to reflect the changes in the urban environment on time. The Internet of Things (IoT) enables real-time data collection to capture these changes. Combined with 3D city models, IoT allows the interactive visualisation of patterns and trends in UDTs. In this study, we conduct investigations on the requirements for the web visualisation of semantic 3D city models enriched with time-dependent properties from IoT and simulation data. We explore the 3D models and IoT data integration requirements, 4D web visualisation design considerations, and the technical implementation requirements for rendering dynamic properties for UDTs applications. The paper also presents a workflow and a web viewer prototype for the 4D visualisation of integrated 3D models and dynamic data.

Keywords: IoT, 3D Models, Urban Digital Twins, 4D Visualisation

1 Introduction

The estimated global population in urban areas is 58% and is expected to rise to 68% by 2050. The rate is even higher in Europe at 76% and on an upwards trend to reach 83% by 2050 [1]. As a result, cities face complex urbanisation challenges, especially concerning air quality, urban mobility, waste management and security. Smart City initiatives rely on digital technologies, especially the Internet of Things (IoT) and digital 3D models of city objects, to address these challenges and make city operations more efficient and liveable for the benefit of their inhabitants.

A digital twin represents a physical object, system, or process in a virtual world, created using models of the defined entities and dynamic data from sensors or simulations. Specifically, Urban Digital Twins (UDTs) are virtual representations of physical urban environments created by integrating 3D city models and IoT sensor data. 3D models describe the built city environment, while IoT sensors or simulation results feed the GIS-based 3D models with time-dependent data. IoT not only enables sensing the

environment but also controlling the communication of the changes in the digital world to the physical world using actuators.

Integrating 3D city models and dynamic data establishes links between modelled physical entities and their respective dynamic data sources. The fourth temporal dimension from the integration enables the 4D visualisation of metrics from real-time and historical dynamic data using game engines or web browsers. Users primarily interact with UDTs through visual tools to understand dynamic urban environments in smart city applications such as monitoring of city operations, urban analytics, and simulations. Web visualisation has an edge over game engines since it requires no extra software installation or hardware by the user apart from a web browser. A large base of city stakeholders can thus interact with UDTs through 4D web visualisation.

This paper investigates the various requirements for effective 4D web visualisation of UDTs. We focus on (1) Requirements for efficient web visualisation of 3D semantic models. (2) 3D city models and dynamic data integration requirements. (3) Dynamic data harmonisation requirements. (4) The cartographic and user requirements to visualise dynamic data effectively and interactively in 4D web maps.

2 Background and Related Research

2.1 Integration of 3D City Models and IoT Data

The first steps towards integrating heterogenous datasets in UDTs are standardisation and harmonisation to standard formats and data models. Semantic 3D city models, which form the basis for developing UDTs, represent buildings, vegetation, city furniture, transport networks, and structures. The Open Geospatial Consortium (OGC) CityGML [2], [3] is the most common, open standard for modelling, storing, and exchanging semantic 3D models in the geospatial domain. Currently, version 2.0 of the standard is in use, and development of the next-generation version 3.0 has been completed recently. Both commercial and open-source tools support the standard. The 3DCityDB is an open-source software suite with a database schema for relational database management systems (DBMS) and a tool that facilitates loading the database with CityGML data and exporting it in various visualisation formats [4].

On the other hand, the OGC SensorThings API standard provides an open and unified way to interconnect IoT devices and data via the web [5]. The standard defines a data model for managing IoT data and a RESTful API for integration with other web-based services. A vital API feature is the native geo-location support enabling the storage and retrieval of sensor data and the corresponding spatial properties. The API is highly scalable and ideal for large-scale IoT deployments like cities. In Hamburg, the API has been implemented in the urban data platform and supports over 2,000 sensor stations [6]. FIWARE is another open-source platform that provides standardised APIs for managing IoT data in Smart City applications [7]. The platform uses the Next Generation Service Interface (NGSI) data model and API to integrate and manage heterogeneous sensor data.

The need to create dynamic UDTs has led to research efforts on integrating semantic 3D city models and dynamic data from sensors and simulations. CityGML version 3.0

has introduced the Dynamizer module for integrating time-dependent properties with 3D city models using data from sensors and simulations [8], [9]. The Dynamizer, defined as a feature type, stores time-series data and updates time-dependent CityGML feature properties. The module also supports modelling complex patterns that represent time variation of properties based on statistics or simulations. For highly variable dynamic data, such as those from sensors, the module supports linking features to sensor data streams using explicit connection information to sensor data services such as the OGC SensorThings API or FIWARE. [10] used the Dynamizer concept as an Application Domain Extension (ADE) in CityGML 2.0 to enrich and visualise semantic 3D building models with simulation energy consumption data.

The CityThings approach [11] has been proposed for integrating sensor data with 3D city models using OGC standards, OGC SensorThings API, and OGC CityGML. The approach links dynamic data into 3D models by extending SensorThings API's Things entities with the unique identifiers of target CityGML features. Since SensorThings API allows defining an arbitrary number of user-defined properties while creating a Thing entity, unique gml_ids of corresponding CityGML features were added as properties to things. This integration enables fetching sensor data from SensorThings API using the gml_ids as query filters for web-based visualisation.

2.2 4D Web Visualisation in UDTs

Developing 3D Web GIS applications requires converting 3D models into streaming formats and using Web mapping libraries. The CesiumJS open-source JavaScript library by Analytical Graphics Inc offers the capability to create 3D globes and interactive maps that run natively on modern browsers. It allows visualising multiple 2D and 3D spatial dataset formats on a virtual globe. In addition, Cesium has developed two data streaming formats, the OGC 3D Tiles [12] and Cesium Modelling Language (CZML). The 3D Tiles format is designed for streaming and rendering massive 3D geospatial data, while CZML is a JSON format for describing time-dynamic 3D scenes.

Although the 3D Tiles format has no temporal support, some studies have implemented dynamic styling of 3D tiles based on temporal data. One study developed a web application to visualise traffic and air quality data using CesiumJS by dynamically styling 3D Tiles based on user interactions or background sensor data updates [13]. Another study used CesiumJS and 3D Tiles to visualise real-time energy simulation results of buildings integrated into CityGML using EnergyADE [14]. To enable end-users to visualise different simulation scenarios, they configured different 3D tiles styling strategies that users could change interactively. Additionally, the study combined colour and opacity visual variables to show multiple properties of a building simultaneously. The study identifies the need to use non-photorealistic visualisation of 3D city models for tasks such as simulations to speed up rendering on the web.

Another study developed dynamic web visualisation of traffic simulation results and 3D city models [15]. The simulation results were derived from SUMO, a microscopic

traffic simulation tool [16] and are composed of movement data of different categories of traffic participants, pedestrians, and passenger cars. The study sought to create accurate visualisation of the simulation results by addressing three visualisation requirements: sufficient spatiotemporal resolution, models that match the physical environment, and smooth rendering of the visualisations on the web. The SUMO simulation results were converted into a CZML document for visualising traffic participants' movements in a CesiumJS-based web client alongside 3D city models. The study proposed a spatial and temporal tiling strategy for CZML data to improve the rendering performance on the web as an alternative to streaming CZML data incrementally.

2.3 Map Design Principles for 4D Visualisation

UDTs users interact with the physical urban environment through dashboards, 2D and 3D maps, and virtual and augmented reality user interfaces. The interfaces allow visualising the state of the UDTs, and in conjunction with actuators, they can control the physical world. 3D maps provide highly detailed visualisation and offer more insight into how objects relate to each other. Different UDTs users' needs should be considered when designing the interfaces above [17].

By integrating a time component, 3D maps gain a fourth dimension that enables the dynamic visualisation of patterns and trends. The temporal dimension could lead to a high volume of data that might impede the user experience while interacting with the maps. The interaction could be affected by either the degradation of web performance due to large datasets or visual clutter from the extra time-dependent information. The complexity caused by the large datasets could be solved using several strategies. One way is to filter features not required in a visualisation scene using attributes, spatial extent, or time property [18]. The visibility of data layers in a map could be toggled either by user controls or based on map scale or the view position.

In UDTs, non-photorealistic 3D maps are preferred to visualise thematic information using cartographic visual variables. The non-photorealistic visualisation removes non-required graphic details and thus highlights the intended information [19], [20]. The reduction of visual elements reduces the data sizes, which improves the real-time rendering of 3D maps.

Cartographic visual variables refer to how geographic features and data are represented in maps. Commonly used visual variables in 3D mapping include feature size, shape, colour, brightness, hue/saturation, transparency, texture, orientation, and position [21]–[23]. The variables can be modified according to thematic and dynamic data in 4D visualisations. The choice of visual variables and their modifications should conform to users' expectations by depicting the physical environment in a familiar presentation [18], [24].

To visualise complex information in maps, including 3D scenes, a single visual variable often cannot represent complete information about a feature to the users. As discussed in Section 2.2, the study by [14] used a combination of colour and opacity to visualise the simulated energy consumption and the ages of buildings, respectively. In another related qualitative study, [20] used compound visual variables to enhance the visualisation of LOD2 building models. The study introduced a primary variable to

visualise attribute data of buildings and a secondary variable to investigate user cognition of the 3D maps. The main variables used were colour and transparency to represent building populations, while the light source and the background were applied as secondary variables. In the study, changing the background properties of the 3D scenes influenced users' spatial cognition, while changing the light source had little impact on how users interpreted the represented data.

3 Visualisation of Integrated 3D City Models and Dynamic Data in UDTs

3.1 Conceptual Framework

Mapping semantic 3D city models provides a more intuitive visualisation of space than 2D maps. On the other hand, dynamic data from sensors and simulations enable the visualisation and analysis of temporal trends. Integrating the two categories of datasets further enhances the visualisation of dynamic processes in the urban environment. This paper identifies the following requirements for effective 4D web visualisation of integrated 3D city models and dynamic data.

R1: Efficient web representation of large 4D models.
The OGC CityGML provides a standardised way to model, store, and exchange semantic 3D city models but cannot be visualised directly on the web. Therefore, the 3D models must be converted into web streaming formats such as OGC 3D Tiles or OGC Indexed 3d Scene Layer (I3S) [25]. These formats support the visualisation of massive geospatial datasets using 3D web mapping libraries such as CesiumJS and deck.gl1. However, these formats do not natively support temporal data. Some workarounds support 4D web visualisations by updating 3D Tiles' properties and styles on runtime using sensor or simulation data [13], [14].

CesiumJS supports dynamic visualisation using the CZML format for moving objects with time-varying geometries. While 3D Tiles and I3S have tiling strategies for optimising the rendering of massive 3D datasets, large CZML files significantly impact web performance. Optimisation measures such as the proposed spatial and temporal tiling strategy by [15], asynchronous loading of CZML data and simplifying geometries should be used to improve web performance

R2: 3D city models and dynamic data need to be connected.
4D web visualisation applications require understanding the connection between semantic 3D models and dynamic data. The CityGML 3.0 Dynamizers module introduces a concept for linking 3D city objects and their time-dependent properties to dynamic data sources. The linking could be exploited to fetch dynamic data and update spatial, thematic and appearance attributes of 3D city objects in 4D web visualisations. Another

[1] https://deck.gl/

option to establish links is using the CityThings [11] approach, which links SensorThings API Things entities to their related 3D objects. With this approach, a 4D web application could check which sensor data in a scene links to city objects.

R3: Harmonised access to dynamic data.
Dynamic datasets come from heterogeneous IoT sensors and simulation results. The various datasets must be harmonised to ease the integration in 4D visualisations. Standards like the OGC SensorThings API and FIWARE provide the data models and standardised APIs for managing and accessing the datasets. The SensorThings API offers standard REST API with spatial, temporal, and thematic filtering capabilities, which makes it suitable for harmonising heterogeneous data in a single repository. Another approach would be to harmonise heterogeneous dynamic datasets using the InterSensor service [26], which performs on-the-fly encoding of the datasets to either OGC Sensor Observation Service or SensorThings API without storing the datasets.

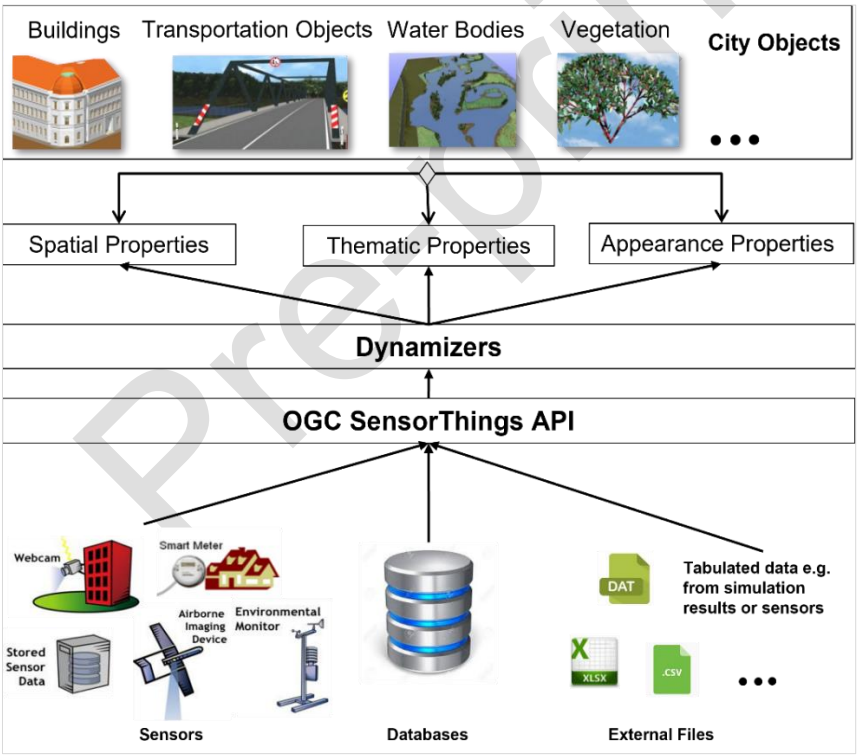

Fig. 1. A concept for standardised management of heterogeneous IoT data and simulation results using SensorThings API. The standardised dynamic data is further integrated with 3D city models using CityGML 3.0 Dynamizers. Adapted from [8]

Based on requirements R2 and R3, streamlined integration of dynamic data into 3D models, for instance, could be performed in two steps using OGC standards: standardisation using SensorThings API and linking 3D models' dynamic properties to sensor data in the SensorThings API using CityGML Dynamizers. This integration, as shown in **Fig. 1**, would enable real-time updates of time-dependent spatial, thematic and appearance attributes of 3D models. By harmonising sensor and simulation datasets into the same data model, dynamic data integration into 3D models using Dynamizers is standardised. The 3D city models may contain zero to many dynamic properties; therefore, the Dynamizers must link each time-dependent property to the respective data stream in SensorThings API.

R4: 4D Maps Design and User Requirements
4D web maps require selecting appropriate cartographic variables and styles that visualise the physical world and its dynamic processes in formats that are familiar to users and easy to interpret. The choice of visual variables depends on data characteristics and the amount to be visualised in a scene. A combination of variables may be required to visualise complex data or the same data at different levels. Visual variables for dynamic data could be applied by changing the appearance, geometry, or project information on 3D objects depending on the data type and 3D objects. Therefore, most visual styling must be determined either manually or automatically.

Web map design should consider different user and application needs to realise the benefits of integrated visualisation. For instance, domain experts require deep insights from the integrated datasets, while decision-makers will be interested in higher-level indicators. Visualisation design should thus consider the information load needed by each group of users and provide tools for data filtering, toggling the visibility of layers, and interacting with the datasets.

The following sections present a workflow and a web viewer prototype for the 4D visualisation of integrated 3D models and dynamic data. We conducted our experiment using Boulevard Sonnenstrasse, a street in the centre of Munich City, as the study area.

3.2 Datasets

GIS Datasets
We use three spatial datasets to represent the Boulevard Sonnenstrasse area on 3D web maps. First, we have LOD2 3D building models in CityGML format, provided as open data by the state mapping agency of Bavaria Munich. The second dataset is a street space model which offers a polygonal representation of the different traffic lane classes and non-traffic areas. Additionally, there is a road markers dataset. The map in **Fig. 2** shows the lane model and the road markings, which were acquired in shapefile format.

Fig. 2. A map of Boulevard Sonnestrasse showing the street space model, inductive loop detectors and the air quality monitoring station.

Sensor and Simulation Datasets

The second set of datasets comes from actual sensor data and simulation results. Traffic data from Inductive Loop Detectors (ILD) in Munich is obtained from Mobilitäts Daten Marktplatz (MDM)2 platform, a German portal for sharing mobility data. The ILD dynamic data for each measured lane contains traffic speeds, flows and occupancy retrieved at 15-minute intervals. Some of the ILDs in our study area are shown on the map in **Fig. 2**. We also fetch traffic lights switching data from the MDM platform, which we have not yet used in the visualisation experiments. The mobility department in Munich City shares the traffic datasets in DATEX II3 format, an XML-based standard for exchanging traffic data.

We also retrieve air quality data from measuring stations in Munich operated by (Bayerische Landesamt für Umwelt (LfU). The stations provide hourly measurements of different air quality parameters concentrations; Air Quality Index (AQI), Particulate Matter (PM10), Carbon Monoxide (CO), Ozone (O3), Sulphur Dioxide (SO2), Nitrogen Dioxide (NO2). We obtained the dataset from the German Environment Agency's (Umweltbundesamt)4 REST API. In the map shown in **Fig. 2** the München/Stachus station that lies within the study area is shown.

In addition to the actual sensor data, we use traffic simulation results from the microscopic traffic simulation tool SUMO. The simulated data includes data from ILDs

[2] https://www.mdm-portal.de/
[3] https://www.datex2.eu/
[4] https://www.umweltbundesamt.de/api

in the SUMO network, which contains the same information as actual ILDs in the city's road network, and the simulated vehicles' Floating Car Data (FCD).

Data Standardisation and Preparation

The spatial and sensor datasets discussed in the previous sections come from different sources and have various data models. We transformed the datasets to OGC standards to ensure data consistency, interoperability, and easy transferability to other cities. The traffic and air quality sensor datasets were integrated into the OGC SensorThings API, while the spatial datasets were converted into OGC CityGML format. The following sub-sections detail the transformation processes shown in **Fig. 3**.

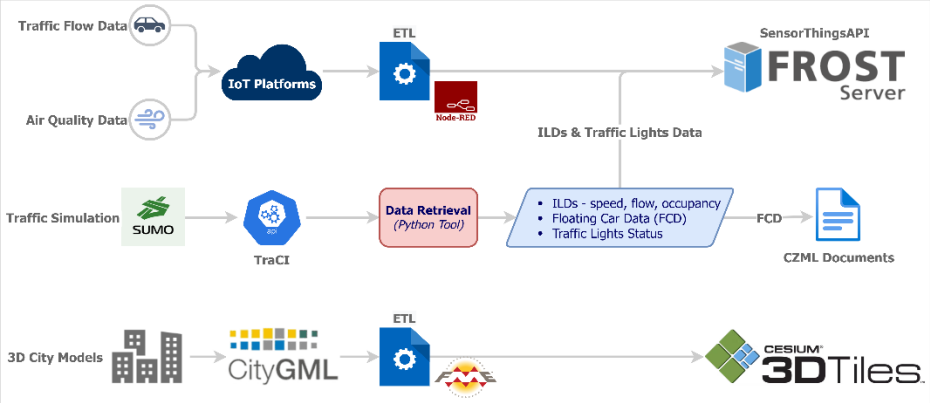

Fig. 3. Standardisation workflow for heterogenous 3D city models and sensor data using OGC standards.

Sensor Data

The sensor datasets were transformed using Node-RED[5], an open-source visual programming tool for integrating IoT data and automating data processing tasks. For the ILD data, we get two sets of data. The first dataset contains the static list of all ILDs, with their IDs and location coordinates. From this dataset, we created entities following the SensorThings API data model. The entities for each ILD included a *Thing*, its *Location*, and a *Datastream* for each *ObservedProperty*. Before creating the Datastreams, we had created the three *Observed Properties*, speed, flow and occupancy, and their *Sensors*. The second dataset contains the temporal *Observations* from the ILDs. We used Node-RED to retrieve and store the data periodically by matching the *Observations* to their respective *Datastream* in the SensorThings API server.

For the air quality data, we created a *Thing*, *Location*, and *Datastreams* for each measurement station for the air quality parameters; *AQI, PM10, CO, O3, SO2, and NO2*. The sensor *Observations* were retrieved at hourly intervals and stored in the created *Datastreams.*

[5] https://nodered.org/

Simulation Data

We developed a Python tool to fetch SUMO simulation results using the Traffic Control Interface (TraCI) library [27]. The connection allows retrieval of movement data from all traffic participants, ILDs, and traffic lights. However, our experiments' target was the Floating Car Data (FCD) from the passenger vehicles category. Still, one could subscribe to all traffic participants, including bicycles, motorcycles, pedestrians, buses, trucks, and trams.

The next step was to define the specific variables we wanted to retrieve for each vehicle at each simulation step. These were the vehicle's position in geographic coordinates (x,y,z), angle, slope, time and speed.

The next step was to create a CZML file to store each vehicle's timestamps, positions, and orientations in the running simulation. The CZML file structure is made up of two main parts. The first part is a header object with an ID, name, CZML version, and clock property. The clock property is an object that specifies the period covered by the file and the starting time of the animation. In the second part, individual vehicles were added as objects called packets. Each packet contains the ID, name, description, optional 3D model path, position in geographic coordinates, and the orientation of the 3D model defined in unit quaternions at each time step. Unit quaternions are mathematical notations for representing spatial orientations in 3D space and are widely used in animation and simulation applications. A similar study by [15] details the workflow required to convert SUMO simulation results into CZML format. We adopted the process in our Python tool to calculate the orientation of vehicle 3D models using positional coordinates, angles, and slopes. The CZML structure is extensible, with the option to add custom properties. We included the vehicle speed at each time step as a custom property. **Fig. 4** shows a snapshot of the created CZML dataset.

```
[
  {"id": "simulation001",
    "version": "1.0",
    "name": "SUMOTrafficSimulationOutput",
    "clock": {"interval": "2023-04-03T07:00:00Z/2023-03-31T07:10:00Z",
              "currentTime": "2023-04-03T07:00:00Z",
              "multiplier": 1}
  },
  {"id": "18444",
    "name": "passenger",
    "model": {"gltf": "./3Dmodels/CesiumMilkTruck.glb"},
    "position": {"epoch": "2023-04-03T07:00:00Z",
                 "cartographicDegrees": [ 44.0,  11.56826, 48.140172,0,
                                          45.0, 11.568173, 48.140219,0] },
    "orientation": {"epoch": "2023-04-03T07:00:00Z",
                    "unitQuaternion": [  44.0, 0.3386128257985951, -0.1138048884613862,0.8076963287179619,-0.4690591030376744,
                                         45.0, 0.33195797886865874,-0.13196145270335846,0.7811314654557384,-0.5120778348462516  ]},
    "properties": { "speed": {"epoch": "2023-04-03T07:00:00Z",
                              "number": [44.0,22.5,
                                         45.0, 21.2]}}
  },
  {"id": "18445",...

  }
]
```

Fig. 4. A snippet of the CZML document created from SUMO traffic simulation results.

GIS Data

The LOD2 3D building models in CityGML 2.0 format were converted using FME into 3D Tiles format. For the semantic street space model and the road markings, we first converted the datasets from shapefile format into CityGML and then into 3D Tiles using the FME software. The street space model conversion follows the CityGML 3.0 Transportation Model, allowing a detailed representation of traffic spaces [28].

Dynamic 3D Models

Beyond visualising the static semantic properties of the 3D city models, our experiment explores adding time-dependent properties to the 3D models to show their temporal variation. The driving lanes from the street space model in shapefile format were selected and extracted for this experiment. The traffic states at each driving lane are dynamic and could be updated by fetching real-time data from ILDs installed in the lanes. We needed, therefore, to establish links between the dynamic properties of each lane and the sensor data measured by ILDs and stored in our SensorThings API server.

One limitation of our dataset is that not all lanes had ILDs installed. The lanes with no ILDs linked would lead to non-smooth visualisation of dynamic properties. For complete coverage of the traffic space with dynamic data, we currently make the simplifying assumption that the traffic state of a lane without a linked ILD is the same as that of a preceding lane.

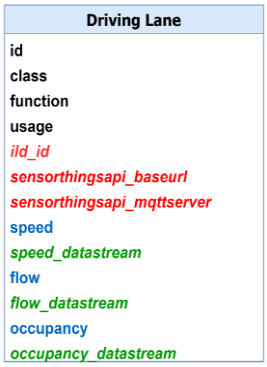

Fig. 5. A simplified attribute data table structure of the driving lanes datasets. The fields in red contain SensorThings API HTTP and MQTT URLs. The blue fields indicate the dynamic properties, while the green fields store the SensorThings API *Datastream* IDs for each dynamic property.

The next step in the process was to add links to the ILDs data in SensorThings API. For this, we use a Python script to extract Datastream IDs for the ILDs that belong to the lanes and create a CSV file. Three new columns were added for each traffic state descriptor in the driving lanes datasets: speed, flow, and occupancy. Two more fields for the SensorThings API server HTTP and MQTT URL addresses were added and populated with the respective links. In the final step, we extended the driving lanes attribute by adding Datastream IDs from the CSV file through a spatial join based on the ILD

ID field. **Fig. 5** shows a simplified view of the attributes table structure after processing. Once we extended the driving lanes dataset with the dynamic traffic state descriptor fields, we converted the dataset from the shapefile format into CityGML 3.0 and then to 3D Tiles format for visualisation.

3.3 Experiment Implementation

UDT Web Viewer Prototype

A UDT web viewer prototype was developed to visualise the semantic 3D models and the integrated dynamic data from traffic sensors. The application was built using *React*[6], a popular modern JavaScript framework for creating user interfaces. In addition to *CesiumJS*, the following JavaScript libraries were used:

1. *Resium*[7] - a library for creating React components for Cesium.
2. *MQTT.js*[8]- a client library for the MQTT protocol on browser-based environments.
3. *Redux* [9]– a state management library for JavaScript applications.

Datasets Configuration

In this step, we created a layer's configuration file that defined how different datasets would be loaded and handled in the application. For the 3D Tiles, we create an object array that holds the different datasets. **Fig. 6** shows an example of an object defining the Traffic layer created using the driving lanes data.

```
{   layerID: 'Traffic',
      url: process.env.PUBLIC_URL + "/geodata/TrafficFlow/tileset.json",
      featureID: 'id',
      icon: <TrafficRounded fontSize="medium" />,
      show: true,
      heightOffset: 0.3,
      dynamicProperties: [
          { property: "flow", datastream: "flow_datastream", description: "Traffic Flow", symbol: '',
            sensorthingsapiBaseUrl: 'https://sta/frost/v1.1', sensorthingsapiMQTT: 'mqtt://sta:8883', },

          { property: "occupancy", datastream: "occupancy_datastream", description: "Traffic Occupancy", symbol: '%',
            sensorthingsapiBaseUrl: 'https://sta/frost/v1.1', sensorthingsapiMQTT: 'mqtt://sta:8883', },

          { property: "speed", datastream: "speed_datastream", description: "Traffic Speed", symbol: 'km/h',
            sensorthingsapiBaseUrl: 'https://sta/frost/v1.1', sensorthingsapiMQTT: 'mqtt://sta:8883', }
          ],
      style: new Cesium3DTileStyle({
                  defines: { occupancy: "${feature['occupancy']}"},
                  color: { conditions: [
                                          ["${occupancy} > 15", "color('#FF0000', 0.2)"],
                                          ["${occupancy} > 10", "color('#FFFF00', 0.2)"],
                                          ["${occupancy} >= 0.1", "color('#00FF00', 0.2)"],
                                          ["true", "color('#ffffff', 0.1)"],],},
                                  }),
  },
```

Fig. 6. 3D Tiles layer configuration object.

[6] https://react.dev/
[7] https://resium.reearth.io/
[8] https://github.com/mqttjs/MQTT.js
[9] https://react-redux.js.org/

The properties of the object in **Fig. 6** are described in **Table 1**. The only mandatory parameters in the configuration object are *layerID*, *URL*, and *featureId.*

Table 1. Description of the configuration layer parameters.

Property	Description
layerID	An ID used to manage the data layer in the application
URL	Path to the 3D Tiles data
featureID	Feature attribute used for selecting and picking
icon	An icon for the layers' navigation panel
show	Indicates whether the layer will be displayed by default
heightOffset	Applies a height offset to 3D Tiles in metres
dynamicProperties	A *Dynamizer* object for layers with features containing dynamic properties.
style	Defines the styling rules based on Cesium's 3D styling language.

The *dynamicProperties* configuration object links 3D feature properties to their associated SensorThings API *Datastreams* by implementing the CityGML Dynamizer concept in the web visualisation workflow. 3D Tiles, by default, do not support temporal data, but when loaded in *Cesium as Cesium3DTileset* layers, their properties can be updated using the *setProperty* function. We define the styling rules based on a dynamic property to visualise temporal data. The workflow for updating the dynamic properties is described in the following section.

Process Flow

Once the 3D Tiles are loaded as *Cesium3DTileset* objects in the web application, the individual features are iterated and stored in the application's state. The application fetches the sensor data from the SensorThings API server using the base URLs and Datastream IDs specified in the configuration file for features containing dynamic properties. Sensor observations are fetched for each feature using a dynamic URL containing time filters for a predefined period and stored in the application's state. The data is fetched asynchronously to allow other processes to run in parallel. If a height offset is specified in the configuration object, the 3D Tiles layer elevation is adjusted in metres to avoid overlap with other layers.

Once the data is stored, a custom function filters the data based on the current Cesium clock time. It updates the corresponding 3D tile feature with the dynamic property or properties from the filtered dataset. For each feature with a predefined style based on a dynamic property, the appearance updates according to the new property value. The diagram in **Fig. 7** shows the process flow for visualising 3D Tiles with dynamic properties.

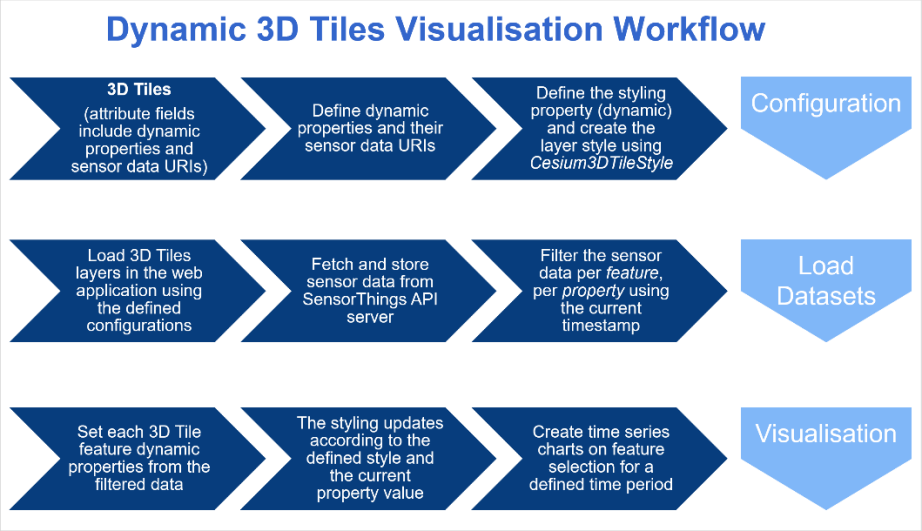

Fig. 7. Web application process flow for visualising 3D city models in 3D Tiles format and their dynamic properties.

User Interface and Interaction

The web application's user interface provides the functionalities required to interact with the datasets. The default *Cesium Timeline* widget controls and displays the scene time. The widget connects to the application's state, so time selection triggers property updates on 3D features with dynamic properties.

A menu button opens a navigation panel that allows users to toggle layer visibility. In the same panel, users can select a time range for the scene by specifying the start and end timestamps. Updating the timestamps triggers an update of sensor data in the background, and the *Cesium Timeline* is updated, too, to reflect the changes.

The application shows a custom information box with a panel of two tabs for features with dynamic properties. The first tab shows a table with the current dynamic property values and time series charts for each property for the entire time range. The second tab shows all the semantic properties of a feature. For features with no dynamic properties, only one tab with a table of properties is displayed. We have used two visual variables for the driving lanes' traffic flow: colour and transparency. The colour variable indicates the level of occupancy levels, while the transparency, in combination with the height offset, allows visualising the underlying street model and road markings. The application demonstrates a 4D visualisation workflow by dynamically updating 3D Tiles properties. By using the *Cesium Timeline* to update the 3D Tiles' properties, additional datasets, such as CZML, could also be visualised simultaneously. **Fig. 8** shows a screenshot of the viewer.

Fig. 8. The user interface for the prototyped UDT web viewer. The driving lanes' traffic occupancy levels retrieved from the SensorThings API are visualised overlaying the street space model.

4 Discussion

4.1 Data Integration Requirements

The first step towards visualising temporal changes in UDTs is integrating 3D city models with sensor and simulation data. Section 2.1. discusses the different approaches used for data integration. Sensor and simulation data could be integrated into 3D models as attributes or fetched dynamically from sensor data repositories. Combining sensor data into 3D models in advance is not feasible as the data is highly dynamic and often large depending on the observed period. Strategies that create explicit links between 3D models and sensor data are thus preferred. The CityGML 3.0 Dynamizer module could further exploit the 3D city models and sensor relationships to determine the appropriate visual variables for 4D visualisations.

4.2 Technical Requirements

The data formats' capabilities and limitations determine their applicability in visualising time-dependent properties in UDTs. Our experiment used the 3D Tiles format for stationary 3D objects and the CZML format to represent moving objects in the CesiumJS-based viewer. Further visualisation types may be required based on the 3D models and sensor data categories. Different chart types, text labels, map icons, and markers could be used to visualise the dynamic properties of 3D models.

Visualising 3D graphics on the web comes at a performance cost due to the high computing power and memory required to load 4D datasets. Adding sensor data increases the data load and computing resources demand. This could result to slow load times and lagging while navigating through a 3D scene. Data fetching, caching, and rendering optimisation strategies are thus critical to improving the performance of UDTs' web applications.

4.3 User Experience Requirements

The needs of end users who interact with UDTs through user interfaces must be met while developing dynamic visualisations. The user interfaces should be easy to navigate, highlight important information, and present data in a visualisation format consistent with user expectations. To further enhance the user experience when portraying 4D web scenes, the choice and placement of visual variables should consider the user's perspective and orientation of 3D models. Different visual variables could be used to display the same dynamic property of a 3D model, depending on the zoom level or camera position.

5 Conclusion and Future Work

This study has investigated different requirements for visualising dynamic UDTs and presented a workflow for developing a web viewer. As discussed in the previous chapter, 4D web visualisation design must consider the data integration, technical, and user requirements. While visualisation requirements and approach focus on semantic 3D city models and sensor data in this study, they also apply to visualising dynamic properties in Building Information Models (BIM).

In the future, more conceptual research is required to extend the data models of the 3D Tiles and I3S visualisation formats to support temporal data and spatiotemporal tiling. As web rendering performance is critical to smooth 4D visualisations, more research on 4D tiling strategies is required. Further technical development is needed to interpret CityGML Dynamizers contained within 3D city models and automatically derive corresponding data structures linking sensor services with the 3D objects used within the 4D web viewer. Above, more basic research will be required regarding the question of good ways to incorporate dynamic data visualisation into 3D scenes. How to leverage the semantic information provided in the 3D city model and different kinds of dynamic data streams? This information should be used to decide, e.g., the style, the usage of visual variables and the 3D/4D placement of visualisations of time series data.

Acknowledgements

We thank the City of Munich for the cooperation in the Connected Urban Twins (CUT) project funded by the Federal Ministry for Housing, Urban Development and Building of Germany. We also thank the city's Geodata Service and Mobility departments for providing datasets used in this study.

References

1. UN-Habitat, 'World cities report 2022: envisaging the future of cities', United Nations Human Settlements Programme (UN-Habitat), Nairobi, Kenya, 2022. [Online]. Available: https://unhabitat.org/sites/default/files/2022/06/wcr_2022.pdf
2. G. Gröger, T. H. Kolbe, C. Nagel, and K.-H. Häfele, 'OGC City Geography Markup Language (CityGML) encoding standard'. 2012.
3. T. H. Kolbe, 'Representing and exchanging 3D city models with CityGML', in *3D Geo-Information Sciences*, J. Lee and S. Zlatanova, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 15–31. doi: 10.1007/978-3-540-87395-2_2.
4. Z. Yao *et al.*, '3DCityDB - a 3D geodatabase solution for the management, analysis, and visualization of semantic 3D city models based on CityGML', *Open Geospatial Data, Software and Standards*, vol. 3, no. 1, p. 5, May 2018, doi: 10.1186/s40965-018-0046-7.
5. S. Liang, C.-Y. Huang, and T. Khalafbeigi, 'OGC SensorThings API Part 1: Sensing', 2016.
6. [M. Fischer, P. Gras, S. Löwa, and S. Schuhart, 'Urban Data Platform Hamburg: Integration von Echtzeit IoT-Daten mittels SensorThings API', *ZFV - Zeitschrift fur Geodasie, Geoinformation und Landmanagement*, vol. 1, p. 47, Feb. 2021, doi: 10.12902/zfv-0330-2020.
7. F. Cirillo, G. Solmaz, E. L. Berz, M. Bauer, B. Cheng, and E. Kovacs, 'A Standard-based Open Source IoT Platform: FIWARE', *IEEE Internet Things M.*, vol. 2, no. 3, pp. 12–18, Sep. 2019, doi: 10.1109/IOTM.0001.1800022.
8. K. Chaturvedi, 'Integration and management of time-dependent properties with semantic 3D city models', Doctoral dissertation, Technische Universität München, 2021. [Online]. Available: https://mediatum.ub.tum.de/?id=1542959
9. T. Kutzner, K. Chaturvedi, and T. H. Kolbe, 'CityGML 3.0: new functions open up new applications', *PFG*, vol. 88, no. 1, pp. 43–61, Feb. 2020, doi: 10.1007/s41064-020-00095-z.
10. E. Chatzinikolaou, I. Pispidikis, and E. Dimopoulou, 'A semantically enriched and web-based 3d energy model visualization and retrieval for smart building implementation using CityGML and Dynamizer ADE', *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. VI-4/W1-2020, pp. 53–60, Sep. 2020, doi: 10.5194/isprs-annals-VI-4-W1-2020-53-2020.
11. T. Santhanavanich and V. Coors, 'CityThings: An integration of the dynamic sensor data to the 3D city model', *Environment and Planning B: Urban Analytics and City Science*, vol. 48, no. 3, pp. 417–432, Mar. 2021, doi: 10.1177/2399808320983000.
12. P. Cozzi, S. Lilley, and G. Getz, '3D Tiles specification 1.0'. Open Geospatial Consortium, 2019. [Online]. Available: https://docs.ogc.org/cs/18-053r2/18-053r2.html
13. H. Ebrahim, T. Santhanavanich, P. Wuerstle, and V. Coors, 'Concept and evaluation of an urban platform for interactive visual analytics', *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci.*, vol. VIII-4/W1-2021, pp. 33–40, Sep. 2021, doi: 10.5194/isprs-annals-VIII-4-W1-2021-33-2021.

14. B. Mao, Y. Ban, and B. Laumert, ‘Dynamic online 3D visualization framework for real-time energy simulation based on 3D Tiles’, *ISPRS International Journal of Geo-Information*, vol. 9, no. 3, p. 166, Mar. 2020, doi: 10.3390/ijgi9030166.
15. C. Beil, M. Kendir, R. Ruhdorfer, and T. H. Kolbe, ‘Dynamic and web-based 4D visualization of streetspace activities derived from traffic simulations and semantic 3D city models’, in *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Copernicus GmbH, Oct. 2022, pp. 29–36. doi: 10.5194/isprs-annals-X-4-W2-2022-29-2022.
16. P. Alvarez Lopez *et al.*, ‘Microscopic traffic simulation using SUMO’, in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, Maui, USA: IEEE, Nov. 2018, pp. 2575–2582. Accessed: May 02, 2023. [Online]. Available: https://www.itsc2019.org/
17. J. Ferré-Bigorra, M. Casals, and M. Gangolells, ‘The adoption of urban digital twins’, *Cities*, vol. 131, p. 103905, Dec. 2022, doi: 10.1016/j.cities.2022.103905.
18. M. Gaigg, *Designing map interfaces: patterns for building effective map apps*, First edition. Redlands: Esri Press, 2023.
19. J. Döllner, H. Buchholz, M. Nienhaus, and F. Kirsch, ‘Illustrative visualization of 3D city models’, presented at the Electronic Imaging 2005, R. F. Erbacher, J. C. Roberts, M. T. Grohn, and K. Borner, Eds., San Jose, CA, Mar. 2005, p. 42. doi: 10.1117/12.587118.
20. B. Li, Z. Luo, and B. Mao, ‘Non-photorealistic visualization of 3D city models using visual variables in virtual reality environments’, *Procedia Computer Science*, vol. 214, pp. 1516–1521, Jan. 2022, doi: 10.1016/j.procs.2022.11.338.
21. J. Bertin, *Graphics and graphic information-processing*. de Gruyter, 1981.
22. F. Hardisty, A. MacEachren, and M. Takatsuka, ‘Cartographic animation in three dimensions: experimenting with the scene graph’, presented at the 20th International Cartographic Conference, Beijing, China: geovista.psu.edu, Aug. 2001. [Online]. Available: https://icaci.org/files/documents/ICC_proceedings/ICC2001/icc2001/file/f17005.pdf
23. V. Rautenbach, S. Coetzee, J. Schiewe, and A. Cöltekin, ‘An assessment of visual variables for the cartographic design of 3D informal settlement models’, presented at the 27th International Cartographic Conference, Rio de Janeiro, Brazil: Maps Connecting the World, Aug. 2015. doi: 10.5167/UZH-117989.
24. J. Döllner, K. Baumann, and H. Buchholz, ‘Virtual 3D City Models as Foundation of Complex Urban Information Spaces’, 2006.
25. C. Reed and T. Belayneh, ‘OGC Indexed 3d Scene Layer (I3S) and Scene Layer Package (*.slpk) Format Community Standard’. Open Geospatial Consortium, 2022. Accessed: Jul. 23, 2023. [Online]. Available: https://docs.ogc.org/cs/17-014r9/17-014r9.html
26. K. Chaturvedi and T. H. Kolbe, ‘InterSensor service: establishing interoperability over heterogeneous sensor observations and platforms for smart cities’, in *2018 IEEE International Smart Cities Conference (ISC2)*, Sep. 2018, pp. 1–8. doi: 10.1109/ISC2.2018.8656984.
27. A. Wegener, M. Piórkowski, M. Raya, H. Hellbrück, S. Fischer, and J.-P. Hubaux, ‘TraCI: an interface for coupling road traffic and network simulators’, in *Proceedings of the 11th communications and networking simulation symposium*, in CNS ’08. New York, NY, USA: Association for Computing Machinery, Apr. 2008, pp. 155–163. doi: 10.1145/1400713.1400740.
28. C. Beil, R. Ruhdorfer, T. Coduro, and T. H. Kolbe, ‘Detailed Streetspace Modelling for Multiple Applications: Discussions on the Proposed CityGML 3.0 Transportation Model’, *IJGI*, vol. 9, no. 10, p. 603, Oct. 2020, doi: 10.3390/ijgi9100603.