INTEGRATED MANAGEMENT AND VISUALIZATION OF STATIC AND DYNAMIC PROPERTIES OF SEMANTIC 3D CITY MODELS

K. Chaturvedi[1,*], Zhihang Yao[2], T. H. Kolbe[1]

[1] Technische Universität München, Chair of Geoinformatics, 80333 Munich, Germany - (kanishk.chaturvedi,thomas.kolbe)@tum.de
[2] virtualcitySYSTEMS GmbH, 85567 Grafing bei München, Germany - zyao@virtualcitysystems.de

Commission VI, WG VI/4

KEY WORDS: Semantic 3D City Models, CityGML, Dynamizers, 3D City Database, Visualization, Timeseries

ABSTRACT:

CityGML is an international standard issued by the Open Geospatial Consortium (OGC) for representing and exchanging Semantic 3D City Models. Due to their large scale and deeply nested structures, the management and visualization of CityGML based models require sophisticated solutions such as the 3D City Database (3DCityDB). The research work presented in this article proposes a high level architecture for extending the 3D City Database to store and manage dynamic properties encoded within a new Application Domain Extension (ADE) of CityGML called Dynamizer ADE. The implementation employs the 3DCityDB 4.2 ADE Plugin Manager, which provides an automatic way for dynamically extending the 3DCityDB to support the storage and management of CityGML models with ADEs. The paper introduces a relational database model for storing and managing the Dynamizer ADE within the 3DCityDB. Further, the research work includes the extension of the 3DCityDB Importer/Exporter in order to import and export CityGML documents including Dynamizer ADE data. 3DCityDB already comes with a Web Feature Service (WFS) interface allowing CityGML features to be requested in standardized ways. The proposed framework enables CityGML Viewers to access static data (using OGC WFS interface) and dynamic data (using the OGC SWE interfaces) in an integrated fashion.

1. INTRODUCTION AND MOTIVATION

CityGML (Gröger et al., 2012) is an international standard issued by the Open Geospatial Consortium (OGC), which allows representing and exchanging Semantic 3D City Models. This standard facilitates the integration of heterogeneous data from multiple sources and allows the representation of the geometrical and semantic attributes of the city level objects (such as buildings, streets, vegetation, and water bodies) along with their interrelationship to other objects. CityGML allows to further decompose complex objects like buildings into their parts like walls, stairs, etc. and these may again consist of parts like windows or doors. Since CityGML objects can have attributes and relations on all levels of this aggregation hierarchy, the exploration of, querying on and interaction with such a 3D city model must also take into account these deeply nested structures. Due to their large scale and deeply nested structures, the management and visualization of CityGML based models require sophisticated solutions.

Numerous software systems have been developed and used for processing and visualizing CityGML data, for example, (FME, 2019), (InfraWorks, 2019), ESRI 3D Cities (Reitz, Schubiger-Banz, 2014), and (azul, 2016). 3D City Database (also known as 3DCityDB) (Yao et al., 2018) is an Open Source software suite, which allows storing, representing, and managing large CityGML datasets on top of spatial relational database management systems (SRDBMS) such as Oracle Spatial and PostgreSQL. It includes a Java front-end application named '3DCityDB Importer/Exporter' for importing and exporting CityGML datasets with arbitrary file sizes. It also allows exporting CityGML objects in the form of 3D visualization formats (such as KML, COLLADA, and glTF) enabling them to be viewed and interactively explored in web applications such as the 3DCityDB Web Map Client or Google Earth. For integration into an OGC Web Service environment, the 3DCityDB provides a Web Feature Service (WFS) interface, using which CityGML objects can be requested in standardized ways. Such software systems and applications allow end users to interact and query with large CityGML datasets in simple and easy ways. For this reason, many cities worldwide such as Berlin, Helsinki, New York, and Singapore are developing and using their city models according to the CityGML standard. There are also numerous applications and simulations benefited using CityGML data (Biljecki et al., 2015).

With its increasing adoption worldwide, CityGML is also being further developed for many new functionalities and extensions. One such extension is the support of time-dynamic properties within CityGML objects. This requirement arises from the fact that many application and simulation scenarios (e.g. environmental simulations, disaster management, traffic simulators) require dealing with dynamic variations of object properties, e.g. variations of (i) thematic attributes such as changes of physical quantities (energy demands, temperature, solar irradiation levels), (ii) spatial properties such as change of a feature's geometry, with respect to shape and location (moving objects), (iii) real-time observations from sensors and IoT devices like Smart Meters. In order to support such time-varying properties within city objects, there have been recent extensions of the CityGML in the form of Application Domain Extensions (ADEs) such as the Energy ADE (Agugiaro et al., 2018) and the Dynamizer ADE (Chaturvedi, Kolbe, 2016). From the application point of view, it is important that existing solutions (databases and visualization applications) dealing

*Corresponding author

with CityGML data support managing and visualizing dynamic properties. However, they only support static properties so far.

This research work provides a novel approach for extending databases and visualization applications for managing and visualizing dynamic properties along with static properties of city objects. The paper proposes a high-level architecture for extending the 3DCityDB to support storage and management of CityGML datasets with a slightly modified Dynamizer ADE (c.f. section 2.1) as a first step. However, a similar approach can also be applied for managing timeseries properties used in other ADEs such as the Energy ADE in the future. The implementation is based on the ADE Plugin Manager (c.f. section 2.2), which provides an automatic way for dynamically extending the 3DCityDB to support storage and management of CityGML models with arbitrary ADEs. The paper introduces a relational database model for storing and managing the Dynamizer ADE within the 3DCityDB. Further, the research work includes the extension of the 3DCityDB Importer/Exporter in order to import and export CityGML documents including Dynamizer ADE data. This enables managing dynamic data (such as time points within timeseries) associated with city objects, which can further be queried and used with standard SQL operations. The high-level architecture also allows accessing and retrieving static and dynamic data in an integrated way. 3DCityDB already contains a Web Feature Service (WFS) interface allowing CityGML features to be requested in standardized ways. The proposed architecture also enables CityGML applications to query and visualize static data (using OGC WFS interface) and dynamic data (using standardized interfaces) in an integrated fashion. For this purpose, the Open Source InterSensor Service (c.f. section 2.3) has been extended to connect to the Dynamizer ADE stored in the 3DCityDB and encode the respective dynamic properties according to the open and international standards.

2. BACKGROUND

2.1 CityGML Dynamizer ADE

Dynamizer (Chaturvedi, Kolbe, 2016) is a new concept, which extends static 3D city models by supporting variations of individual feature properties and associations over time. It provides a data structure to represent dynamic values in different and generic ways. Such dynamic values may be given by (i) tabulation of time/-value pairs using its *AtomicTimeseries* class, (ii) patterns of time/value pairs based on statistical rules using its *CompositeTimeseries* class, and (iii) retrieving observations directly from external sensor/IoT services using its *SensorConnection* class. In addition, Dynamizer delivers a method to enhance static city models by dynamic property values. It references a specific attribute (e.g. geometry, thematic data or appearance) of an object within a 3D city model providing dynamic values overriding the static value of the referenced object attribute. Dynamizers have already been implemented as an Application Domain Extension (ADE) for CityGML 2.0 and are planned to become a part of the next version of CityGML (version 3.0).

The conceptual UML model and XML Instance Schema are already made available for Dynamizer ADE (Chaturvedi, Kolbe, 2017). However, there have been a few recent additions in the Dynamizer UML model which is going to be proposed to the CityGML Working Group. Within Dynamizers, the dynamic data is already modeled as *AbstractTimeseries*, which is responsible for representing time-variant or dynamic values in different and generic ways. The timeseries may be modeled in two ways: (i) *AtomicTimeseries*, and (ii) *CompositeTimeseries*. *AtomicTimeseries* consists of either *dynamicDataDR*, *dynamicDataTVP*, or *observationData*. It allows different representations of timeseries according to the OGC TimeseriesML 1.0 (Tomkins, Lowe, 2016) encodings (interleaved time/value pair and domain-range), and observations encoded according to the Observations&Measurements (O&M) standard (Cox, 2013). O&M is one of the core standards for the response models of OGC Sensor Web Enablement (SWE) (Bröring et al., 2011) based standards such as Sensor Observation Service (SOS) (Bröring et al., 2012) and SensorThings API (Liang et al., 2015). Apart from these two representations, *AtomicTimeseries* now includes two more classes (as shown in figure 1): (i) *GenericTimeseries*, and (ii) *BasicFileTimeseries*. *GenericTimeseries* provides a very basic data structure to represent timeseries data. The advantage with *GenericTimeseries* is that it does not require databases to support complex standards such as TimeseriesML 1.0. However, unlike TimeseriesML, *GenericTimeseries* is not capable to map missing values or multiple values in timeseries using interpolation and aggregation functions. *BasicFileTimeseries* class allows retrieving timeseries from basic external files such as CSV and Excel sheets. Such functionality is helpful in working with scenarios where timeseries data is stored in basic files (e.g. by simulation software). The *BasicFileTimeseries* class provides appropriate metadata for reading/writing timeseries from/to an external file.

2.2 3DCityDB and ADE Plugin Manager

The current release of 3DCityDB (version 4.2) (3DCityDB, 2019) includes a new ADE Plugin Manager for its Importer/Exporter. It allows to dynamically extend a 3DCityDB instance to facilitate the storage and management of arbitrary CityGML ADEs (Yao, Kolbe, 2017). It is implemented based on the Open Source Attributed Graph Grammar (AGG) transformation engine for realizing the automatic transformation from an XML application schema (XSD) to a compact relational database schema (including tables, indexes, and constraints etc.) for a given CityGML ADE. In addition, an XML-based schema mapping file can also be automatically generated which contains the relevant meta-information about the derived database schema as well as the explicit mapping relationships between the source and target schemas and allows developers to implement applications for managing and processing the ADE data contents stored in a 3DCityDB instance. The ADE Plugin Manager has been tested successfully with well-known CityGML ADEs like Energy ADE, Noise ADE, and UtilityNetwork ADE.

2.3 InterSensor Service

InterSensor Service (Chaturvedi, Kolbe, 2019) is an Open Source application, which establishes interoperability over heterogeneous sensor and IoT platforms and other timeseries data in standardized ways. It allows making connections to multiple data sources by using data adapters. These data adapters can be developed to connect to not only different IoT platforms, but also to external databases, CSV files, Cloud-based spreadsheets, GPS feeds, and real-time Twitter feeds. While querying, the service opens a data source

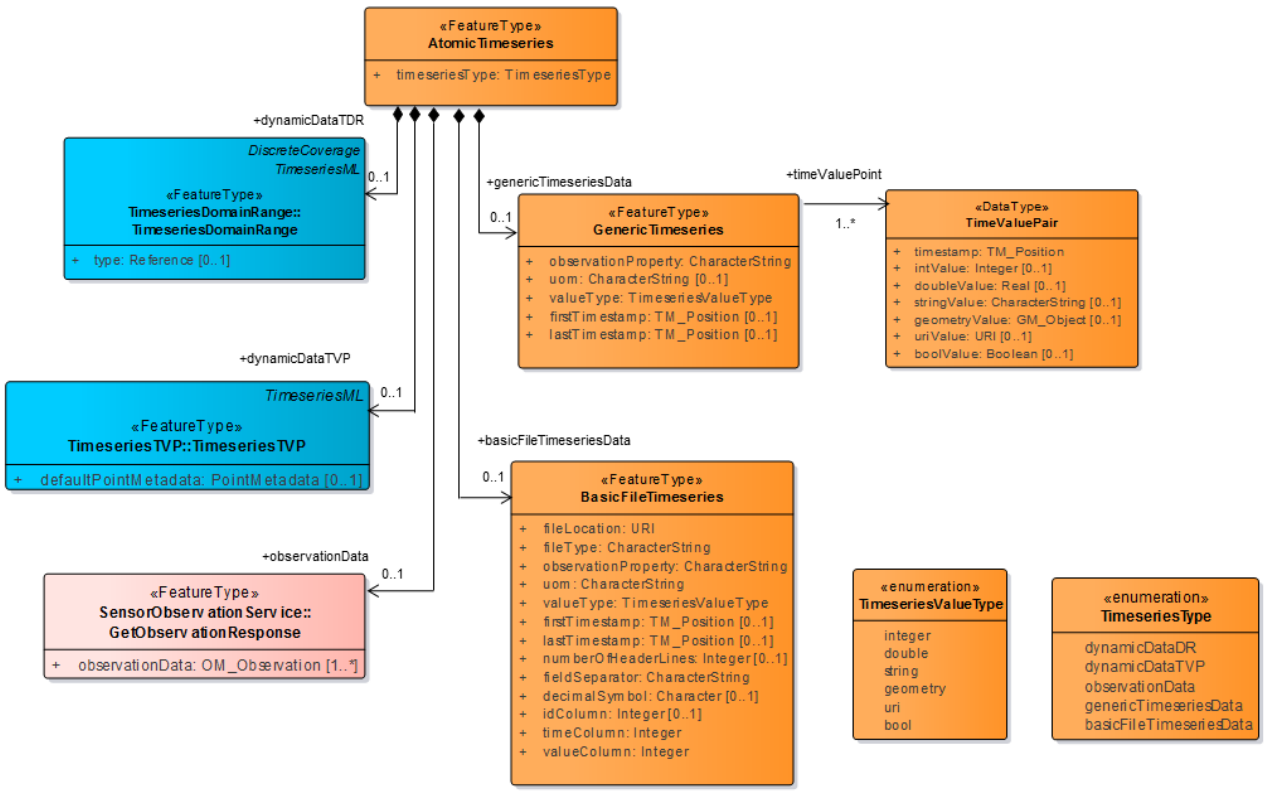

Figure 1. Modified UML Model of the Dynamizer ADE. The AtomicTimeseries includes two new classes: GenericTimeseries and BasicFileTimeseries. The other classes are shown in (Chaturvedi, Kolbe, 2016).

connection and retrieves the observations based on querying parameters directly from the data source. The service encodes these observations "on-the-fly" according to internationally standardized interfaces such as the OGC Sensor Observation Service and OGC SensorThings API. In this way, applications and tools can be developed based on these standards without worrying about what different kinds of sensor platforms they use. Multiple sensors can be attached to these infrastructures and their interfaces will always be common for different applications.

3. EXTENDING THE 3DCITYDB FOR SUPPORTING STATIC AND DYNAMIC PROPERTIES

This section proposes a high level architecture for managing and visualizing the time-dynamic properties along with static properties of Semantic 3D City Models. The architecture extends the 3D City Database (3DCityDB) to store and manage dynamic properties encoded within the CityGML Dynamizer ADE. As shown in figure 2, the 3DCityDB is extended for supporting the Dynamizer ADE. The implementation employs the 3DCityDB ADE Plugin Manager, which provides an automatic way for dynamically extending the 3DCityDB to support the storage and management of CityGML models with ADEs. However, in order to improve querying performance, the relational database model of Dynamizer ADE has been developed by defining three separate modules: (i) Dynamizer core module for storing the core attributes of Dynamizers, (ii) Timeseries Metadata Module (for storing the metadata of Timeseries), and (iii) Timeseries module (for storing time point values). The advantage of keeping the Timeseries module separate from the Dynamizer module is that this approach allows making the Timeseries module re-usable, for example, by storing timeseries from other ADEs such as the Energy ADE. Furthermore, the 3DCityDB Importer/Exporter is extended to facilitate import and export of CityGML documents with Dynamizer ADE data. This enables managing dynamic data (such as time points within timeseries) associated with city objects, which can further be queried and used using standard SQL operations. The framework also allows accessing and retrieving static and dynamic data in an integrated way. 3DCityDB already comes with a Web Feature Service (WFS) interface allowing CityGML features to be requested in standardized ways. However, the WFS is not suitable to query dynamic/timeseries data. The Sensor Web Enablement (SWE) standards provide comprehensive interface models and web services such as the Sensor Observation Service (SOS) and SensorThings API for retrieval of sensor descriptions and observations (i.e. timeseries data) with the help of standardized requests. In comparison to SOS, SensorThings API is a relatively new standard, which is REST-ful, lightweight, and using JSON data encodings. This paper further explains ways to request dynamic data according to the OGC SOS and SensorThings API using the Open Source InterSensor Service. In this way, heterogeneous observations can be analyzed and visualized in a unified way.

3.1 Relational Data Model for Dynamizer ADE

Figure 3 shows the relational database model for the Dynamizer ADE. The following sub sections descrobe the different modules of the relational data model.

3.1.1 Dynamizer Core Module The class *Dynamizer* from the UML model consists of the following core attributes:

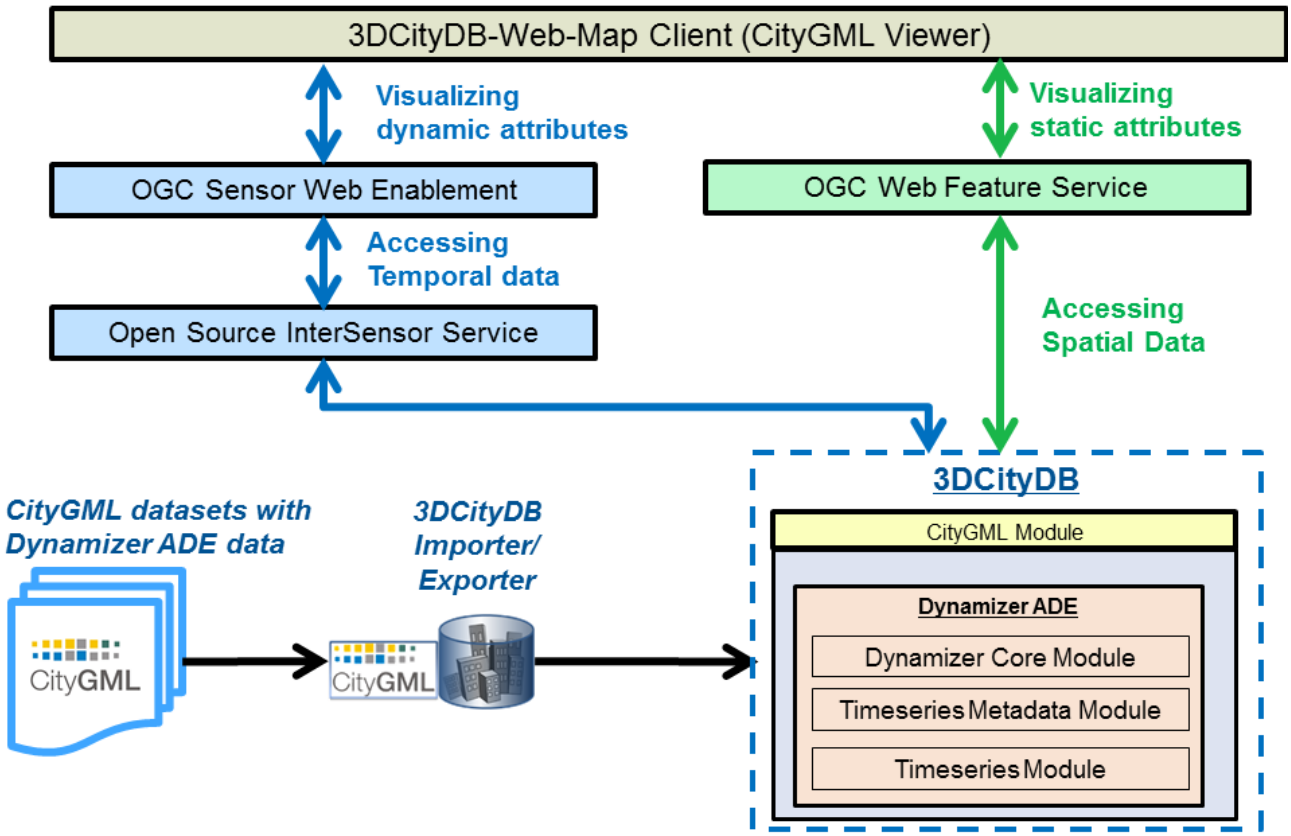

Figure 2. High Level Overview of Managing and Visualizing Static and Dynamic data using CityGML Dynamizers

(i) *Dynamizer_id*, (ii) *attributeRef*, (iii) *startTime*, and (iv) *endTime*. *attributeRef* refers to a specific attribute of a specific city object by using an XPath expression. This is the static property that should be overridden by timeseries data. *startTime* and *endTime* are absolute time points denoting the time span for which the Dynamizer provides dynamic values. These core attributes are stored in the table DYN_DYNAMIZER. In addition, Dynamizer can also provide direct explicit links to external sensor and IoT based services by using the class *SensorConnection*. It includes the following attributes (i) *sensorId* - a unique identification of the sensor/IoT device, (ii) *serviceType* - the type of service such as SOS and SensorThings API, (iii) *sensorLocation* - association with a specific city object that hosts the sensor or to which it is attached, (iv) *linkToObservation* - link to observation response (timeseries), (v) *linkToSensorDescription* - link to the description/metadata of the sensor/IoT device. For example, OGC SOS involves different requests for retrieving sensor descriptions and observations. *DescribeSensor* is used to retrieve the sensor description in the SensorML format. *GetObservation* is used to retrieve sensor observations encoded in the O&M format. The request parameter also allows to include the specification of spatial and temporal filters. Such links can directly be used as a part of the Dynamizer *SensorConnection* without having the need to store the timeseries observations within CityGML. The attributes of the *SensorConnection* class are also stored in the table DYN_DYNAMIZER.

3.1.2 Timeseries Metadata Module Dynamizers also support having timeseries in-line within city objects. The in-line timeseries within Dynamizers can be modeled in two ways: (i) *AtomicTimeseries*, and (ii) *CompositeTimeseries*. As mentioned in section 2.1, *AtomicTimeseries* can be represented according to (i) TimeseriesML Time Value Pair encoding, (ii) TimeseriesML Domain Range encoding, (iii) Observations&Measurements, (iv) Dynamizer Generic Timeseries, and (v) Dynamizer Basic File Timeseries. Based on the type of atomic timeseries, the respective metadata of the timeseries are mapped onto the table DYN_TIMESERIES. For example, if the timeseries is represented according to the TimeseriesML standard and it contains a specific interpolation type, this is stored in the attribute INTERPOLATION_TYPE. Similarly, if the timeseries is being retrieved from an external CSV file, its location is stored in the attribute FILE_LOCATION. The flag IS_ATOMIC is used to determine whether the timeseries is atomic or composite.

In order to manage *CompositeTimeseries*, the database structure is inspired from the existing SURFACE_GEOMETRY table in the 3DCityDB. Since *CompositeTimeseries* compose of an ordered list of *AbstractTimeseries*, several Atomic or Composite Timeseries can be aggregated to form a Composite Timeseries. Each nested timeseries references to its root using the ROOT_ID attribute. This information has a big influence on the query performance, as it allows to avoid recursive queries. If e.g. the retrieval of all timeseries forming a specific composite timeseries is of importance, simply those IDs have to be selected which contain the related PARENT_ID and ROOT_ID. For instance, in energy applications, an Atomic timeseries may be defined for a working day, a Saturday, and a Sunday (represented by A, B, and C respectively). Now, in order to reflect a pattern of energy consumption of an entire week (represented as Wx), a Composite timeseries may contain five repetitions of Atomic timeseries A followed by single representations of timeseries B and C (represented as AAAAABC). Similarly, for reflecting a pattern of the

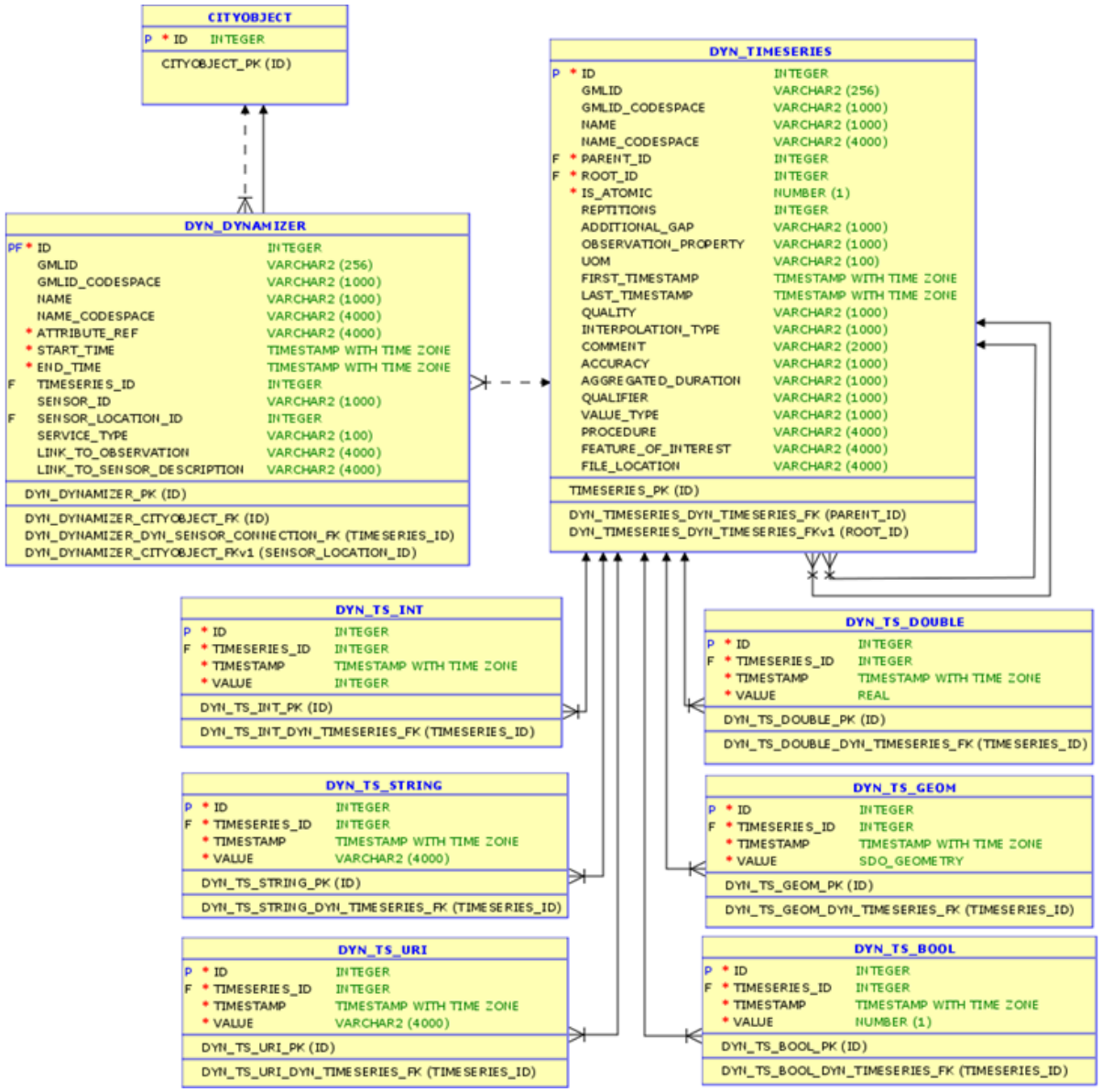

Figure 3. Relational Database Model of Dynamizer ADE. The figure shows individual tables of Dynamizer along with their columns and primary and foreign keys.

entire month (Mx), the Composite Timeseries may contain four representations of the timeseries W (represented as W1,W2,W3,W4). And lastly, for reflecting a pattern of the entire year (Yx), the Composite Timeseries may contain 12 repetitions of the timeseries M (represented as M1,M2,...M12). Hence, in this case, the Timeseries Y would have ID = 1 and ROOT_ID = 1; Timeseries M1 would have ID = 2, PARENT_ID = 1, and ROOT_ID = 1; Timeseries W1 would have ID = 3, PARENT_ID = 2, and ROOT_ID = 1; and so on.

3.1.3 Timeseries Module This modules is responsible for storing the raw timeseries values (time-value pairs). Depending on the source and type, timeseries can be represented according to different data types. For example, a timeseries generated by a weather station for temperature recordings is of datatype *double*, a timeseries from a traffic camera for counting number of cars at a junction is an *integer*, and another timeseries retrieved from a moving GPS a point object.

In order to manage timeseries of different types, 3DCityDB is extended by individual tables: DYN_TS_INT (Timeseries Integer), DYN_TS_DOUBLE (Timeseries Double), DYN_TS_STRING (Timeseries String), DYN_TS_GEOM (Timeseries Geometry), DYN_TS_URI (Timeseries External Link), and DYN_TS_BOOL (Timeseries Boolean),

3.2 Import and Export of the Dynamizer ADE within the 3DCityDB

Once the relational database model is developed, the next step is to extend the Import and Export functionality of the 3DCityDB to map the CityGML documents with the Dynamizer ADE onto the appropriate tables. In order to support the CityGML Dynamizer ADE with the 3DCityDB, three major steps are required to be performed:

1. Mapping the XML Schema definition of the ADE to a relational schema that integrates with the 3DCityDB core schema

2. Creating an XML-based schema mapping file that captures the mapping between elements of the XML schema and elements of the relational schema

3. Registering the ADE with the metadata tables of the 3DCityDB

The recent version of 3DCityDB (v4.2) already provides an ADE Plugin Manager in order to automate these steps. It reads the XML schema and applies a rule-based transformation to derive a relational schema for the ADE that seamlessly integrates with the 3DCityDB. In other words, the ADE Plugin Manager automatically creates the tables and joins based on the classes and their relations defined in the UML model. Users can redefine default rules or even add new rules, and thus have full control over the mapping result. However, as mentioned in section 3.1, in order to improve querying efficiency, the relational database model of the Dynamizer ADE involves only three independent modules: Dynamizer Core Module, Timeseries Metadata Module and Timeseries Module. This approach gives flexibility to re-use existing timeseries modules with other ADEs such as Energy ADE and UtilityNetwork ADE. Hence, the structure of the relational database model is different from the UML model. For this reason, it is proposed to include the Timeseries and Metadata modules as an integrated part of 3DCityDB. However, in the future, the ADE Plugin Manager will be extended to map the Dynamizer UML model in such a way that the Dynamizer core attributes are mapped onto the Dynamizer core table and the associated timeseries are mapped onto the Timeseries and Metadata modules.

Once the ADE is registered with the 3DCityDB by performing the above steps, the 3DCityDB Importer/Exporter tool requires extensions to (i) import timeseries data from Dynamizer ADE to the new Dynamizer ADE tables, and (ii) export timeseries data from Dynamizer ADE tables to the CityGML documents. For this purpose, the 3DCityDB provides the Importer/Exporter tool. Since the Importer/Exporter does not provide generic ADE support yet, the Dynamizer ADE extension is required to be developed against the ADE API of the Importer/Exporter. The Dynamizer ADE extension can be developed by performing the following steps:

1. Creating an ADE module for citygml4j for parsing and writing CityGML with Dynamizer ADE.

2. Implementing the ADEExtension interface of the ADE API and providing Java code for reading and writing data into the ADE tables.

By performing these steps, the functionalities of the Importer/Exporter can be extended for importing and exporting the timeseries data from the CityGML Dynamizer ADE.

3.3 InterSensor Service for Dynamizer ADE

3DCityDB already comes with a Web feature Service implementation for accessing and querying the CityGML features and attributes. However, a WFS is not suitable for querying time-varying data. For that purpose, the Open Source InterSensor Service is utilized to query timeseries values from Dynamizers. The InterSensor Service requires developing a data adapter for 3DCityDB. This data adapter is responsible for establishing a connection to the 3DCityDB tables for the respective Dynamizer IDs using the following parameters:

```
{
  datasource-connection:
    name: "DynamizerConnection"
    description: ""
    connectionType: "Dynamizer"
    databaseType: "PostgreSQL"
    ipAddress: "127.0.0.1"
    port: 5432
    databaseName: "3DCityDB"
    username: "user"
    password: "******"
    dynamizerId: "dyn_01_WS_1_globalRad"
}
```

Using the above parameters, the InterSensor Service forms the appropriate JDBC request and establishes connection to the running 3DCityDB. Based on the provided Dynamizer ID, the InterSensor Service queries the timeseries metadata (such as observation type, unit of measurement etc.) and timeseries data (timestamps and values) can be queried from the respective tables.

In addition, the InterSensor Service also provides external standardized interfaces using which the timeseries data can be queried and visualized using the OGC Sensor Observation Service and SensorThings API. For example, if the InterSensor Service is deployed on the server 127.0.0.1 with port 8080, the Dynamizer timeseries values between a specific time range can be accessed using the SensorThings API standard as follows:

```
http://127.0.0.1:8080/OGCSensorThingsApi/
v1.0/Datastreams(1)/Observations?
$filter=during
(phenomenonTime,2019-01-01T00:00:00/
                2019-07-01T00:00:00)
```

Similarly, the same query on the Dynamizer timeseries can also be performed using the Sensor Observation Service GetObservation request as follows:

```
http://127.0.0.1:8080/ogc-sos-webapp/service?
service=SOS&version=2.0.0&
request=GetObservation&
temporalFilter=om:phenomenonTime,
               2019-01-01T00:00:00/
               2019-07-01T00:00:00
```

Such queries establish connections to the 3DCityDB Dynamizer ADE based on the connection parameters provided in the InterSensor Service. The observations are retrieved directly from the 3DCityDB based on the temporal filter provided in the query. In this way, the timeseries data from Dynamizers can be queried and visualized using the OGC standards.

4. QUERYING AND VISUALIZATION OF DYNAMIC AND STATIC PROPERTIES

Dynamizers have already been implemented as an Application Domain Extension (ADE) within the OGC Future City Pilot Phase 1 (Chaturvedi, Kolbe, 2017). The results are focused on two scenarios: (i) integrating real-time sensor stream with CityGML building models, and (ii) enriching 3D building wall and roof surfaces by time-dependent solar potential simulation

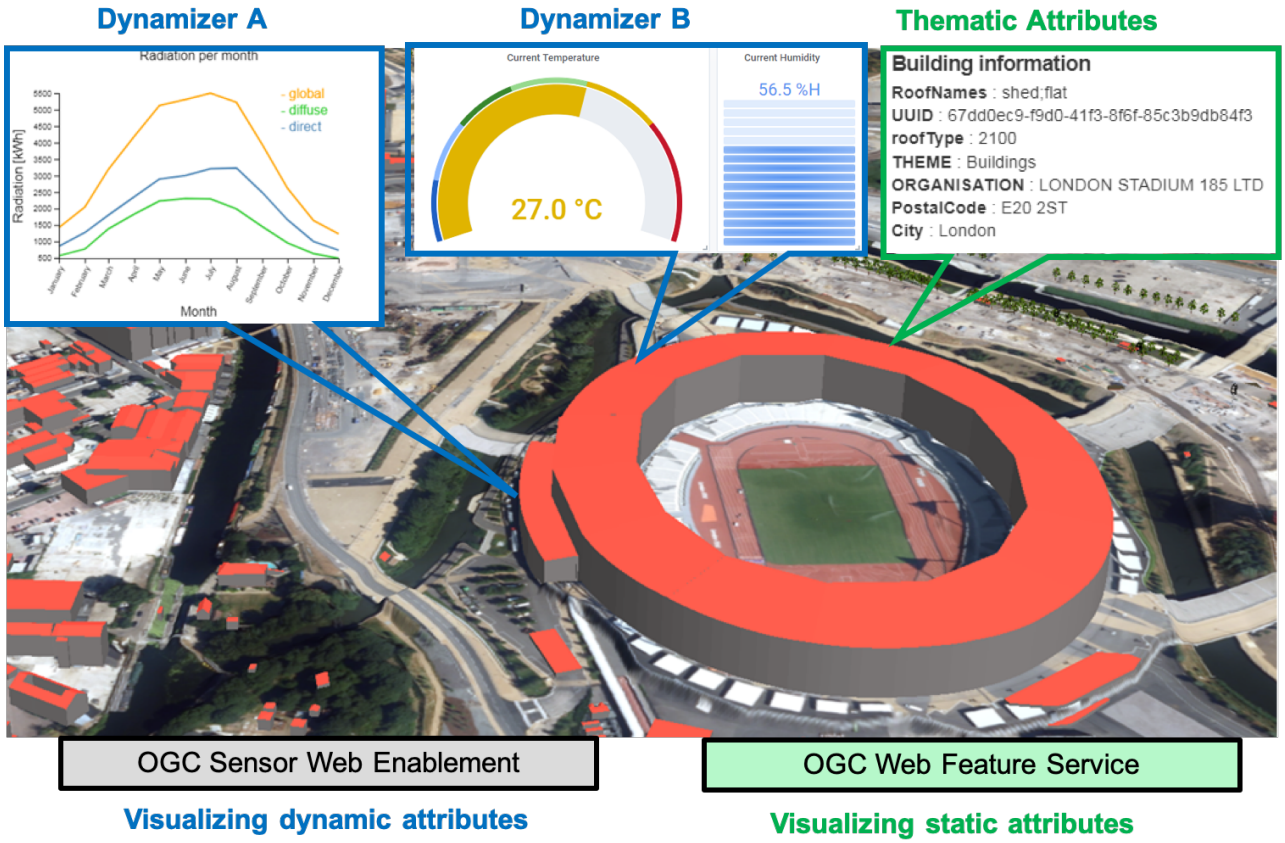

Figure 4. Illustration of an integrated visualization of a building's static thematic attributes and dynamic properties being retrieved from Dynamizer A (solar potential simulation results stored in-line with the Building object) and Dynamizer B (explicit links to the real-time sensor stream).

results. For the first scenario, a real-time temperature and humidity sensor stream is linked explicitly with the CityGML Building object using the Dynamizer *SensorConnection* class. It associates a specific Building property (e.g. a generic attribute called 'temperature') directly with the sensor stream measuring the temperature property for that building and allows overriding the generic attribute according to the result of sensor stream at a specific time. The sensor stream can base on the (i) OGC Sensor Web Enablement standards (such as SensorThings API or Sensor Observation Service), (ii) any IoT platform (such as Thingspeak, OpenSensors, TheThingsNetwork), or some other (proprietary) service. For the second scenario, a CityGML Building object is enriched by performing a solar potential simulation in order to estimate monthly solar energy production for roofs and wall surfaces of buildings. In this scenario, the monthly solar irradiation values are stored using the Dynamizer *AtomicTimeseries* class. The irradiation values are represented as interleaved time-value pairs according to the TimeseriesML standard. Using this approach, one single generic attribute (e.g. directRadiation) of the building wall surface can be overridden according to the monthly values represented within the *AtomicTimeseries*. Moreover, it allows modeling the precise description of timeseries data with its metadata details within the CityGML object. As a result, it makes cross-domain exchanging of simulation results with city objects possible allowing performing detailed realistic simulations. The instance files are available in the Future City Pilot Phase 1 Engineering Report (Chaturvedi, Kolbe, 2017).

The proposed architecture allows importing resulting CityGML Dynamizer datasets into the 3DCityDB. For the scenario 1, the Dynamizer attributes are imported into the table DYN_DYNAMIZER with information about the sensor stream such as its unique ID, links to the sensor description and real-time observations. In this case, since there is no in-line timeseries involved, there is no data import to the Timeseries related tables. For the second scenario, the *AtomicTimeseries* with solar potential simulation results are imported into the Timeseries tables. The metadata such as type of observations and Unit of Measurement are imported into the table DYN_TIMESERIES. Since the simulation results are integer values, the timeseries values are imported into the table DYN_TS_INT. Such management allows performing temporal queries within the database, for example, generating a timeseries graph for direct irradiation values of a building wall surface between March and September of a specific year.

Furthermore, the extensions of the InterSensor Service allow establishing connections to the specific Dynamizers stored in the 3DCityDB and querying and visualizing the dynamic properties using international standards such as OGC SensorThings API and OGC SOS. As shown in figure 4, the static attributes (such as Building geometry and its thematic attributes) can be retrieved and visualized using the Web Feature Service. The dynamic attributes of the same buildings (i) Dynamizer A for direct links to the temperature and humidity sensor, and (ii) Dynamizer B for representing solar potential simulation results of its wall surface, can be retrieved and visualized according to the OGC SWE standards with the help of the InterSensor Service.

5. CONCLUSIONS AND FUTURE WORK

The research work presented in this article extends the 3D City Database to store and manage dynamic properties encoded within the CityGML Dynamizer ADE. However, a similar approach can also be applied for managing timeseries properties within other ADEs such as the Energy ADE in the future. A relational database model for the Dynamizer ADE has been proposed in the paper by defining three separate modules: (i) Dynamizer core module for storing the core attributes of Dynamizers, (ii) Timeseries Metadata Module (for storing the metadata of Timeseries), and (iii) Timeseries module (for storing time point values). The advantage of keeping the Timeseries module separate from the Dynamizer module is that this approach allows making the Timeseries module re-usable, for example, by storing timeseries from other ADEs such as the Energy ADE.

The implementation employs the 3DCityDB 4.2 ADE Plugin Manager, which provides an automatic way for dynamically extending the 3DCityDB to support the storage and management of CityGML models with Application Domain Extensions. Furthermore, the 3DCityDB Importer/Exporter has been extended in order to facilitate the import and export of CityGML documents with Dynamizer ADE data. This enables managing dynamic data (such as time points within timeseries) associated with city objects, which can further be queried and used using standard SQL operations. In the future, the developments will include the creation of Dynamizer ADE module for the citygml4j library.

The framework allows accessing and retrieving static and dynamic data in an integrated way. 3DCityDB already comes with a Web Feature Service (WFS) interface allowing CityGML features to be requested in standardized ways. The proposed framework enables CityGML Viewers to access static data (using OGC WFS interface) and dynamic data (using the OGC SWE interfaces such as SOS and SensorThings API) in an integrated fashion with the help of the InterSensor Service.

REFERENCES

3DCityDB, 2019. 3D City Database for CityGML Version 4.2. https://www.3dcitydb.org/3dcitydb/documentation/ (21 March 2019).

Agugiaro, G., Benner, J., Cipriano, P., Nouvel, R., 2018. The Energy Application Domain Extension for CityGML: enhancing interoperability for urban energy simulations. *Open Geospatial Data, Software and Standards*, 3(1), 2.

azul, 2016. 3D City Model Viewer for MacOS developed by TU Delft. https://github.com/tudelft3d/azul (21 July 2019).

Biljecki, F., Stoter, J., Ledoux, H., Zlatanova, S., Çöltekin, A., 2015. Applications of 3D City Models: State of the Art Review. *ISPRS International Journal of Geo-Information*, 4(4), 2842–2889.

Bröring, A., Echterhoff, J., Jirka, S., Simonis, I., Everding, T., Stasch, C., Liang, S., Lemmens, R., 2011. New Generation Sensor Web Enablement. *Sensors*, 11(3), 2652–2699.

Bröring, A., Stasch, C., Echterhoff, J., 2012. Sensor Observation Service Interface Standard, OGC Doc. No. 12-006. http://www.opengeospatial.org/standards/sos (11 April 2019).

Chaturvedi, K., Kolbe, T. H., 2016. Integrating Dynamic Data and Sensors with Semantic 3D City Models in the context of Smart Cities. *Proceedings of the 11th International 3D Geoinfo Conference*, ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, IV-2/W1, Athens, Greece.

Chaturvedi, K., Kolbe, T. H., 2017. Future City Pilot 1 Engineering Report, OGC Doc. No. 19-098. http://docs.opengeospatial.org/per/16-098.html (11 April 2019).

Chaturvedi, K., Kolbe, T. H., 2019. Towards Establishing Cross-Platform Interoperability for Sensors in Smart Cities. *Sensors*, 19(3).

Cox, S., 2013. Observations and measurements (O&M) | OGC Document No. 10-004r3. http://www.opengeospatial.org/standards/om (21 March 2019).

FME, 2019. Feature Manipulation Engine by Safe Software. https://www.safe.com/ (21 July 2019).

Gröger, G., Kolbe, T. H., Nagel, C., Häfele, K.-H., 2012. City Geography Markup Language (CityGML) v 2.0, OGC Doc. No. 12-019. http://www.opengeospatial.org/standards/citygml (08 March 2019).

InfraWorks, 2019. Infrastructure Design Software by Autodesk. https://www.autodesk.com/products/infraworks/overview (21 July 2019).

Liang, S., Huang, C.-Y., Khalafbeigi, T., 2015. SensorThings API Part 1: Sensing, OGC Doc. No. 15-078r6. https://www.opengeospatial.org/standards/sensorthings (11 April 2019).

Reitz, T., Schubiger-Banz, S., 2014. The Esri 3D city information model. *IOP Conference Series: Earth and Environmental Science*, 18, 012172.

Tomkins, J., Lowe, D., 2016. Timeseries Profile of Observations and Measurements, OGC Document No. 15-043r3. http://www.opengeospatial.org/standards/tsml (15 March 2019).

Yao, Z., Kolbe, T. H., 2017. Dynamically Extending Spatial Databases to support CityGML Application Domain Extensions using Graph Transformations. T. P. Kersten (ed.), *Kulturelles Erbe erfassen und bewahren - Von der Dokumentation zum virtuellen Rundgang, 37. Wissenschaftlich-Technische Jahrestagung der DGPF*, Publikationen der Deutschen Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation (DGPF) e.V., 26, Deutsche Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation e.V., Würzburg, 316–331.

Yao, Z., Nagel, C., Kunde, F., Hudra, G., Willkomm, P., Donaubauer, A., Adolphi, T., Kolbe, T. H., 2018. 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*, 3(1), 5.