CITYGML RESTFUL WEB SERVICE: AUTOMATIC RETRIEVAL OF CITYGML DATA BASED ON THEIR SEMANTICS. PRINCIPLES, GUIDELINES AND BLDG CONCEPTUAL DESIGN

I. Pispidikis *, E. Dimopoulou

School of Rural and Surveying Engineering, National Technical University of Athens, 9 Iroon Polytechneiou str, 15780 Zografou, Greece (pispidikisj@yahoo.gr; efi@survey.ntua.gr)

KEY WORDS: Web Services, CityGML, REST, SOAP, RESTful

ABSTRACT:

CityGML is considered an optimal standard for the representation of 3D city models. However, due to its complex structure, easy-to–use data retrieval is important, in terms of interoperability. This implies choosing the implementation of Web Service Technologies and in particular the WFS, as the most suitable OGC standard for retrieving the real geometry data. Nevertheless, this standard serves data mainly based on their geometry, while CityGML also covers topology and more importantly semantic aspects of 3D city models. Therefore, this paper examines and presents the new CityGML RESTful Web service, instead of the OGC WFS. This Web Service is conceptually designed to achieve CityGML data retrieval based on their semantics characteristics. In this context, several principles and guidelines of the new CityGML RESTful Web service are described and the "CityModels" resource is presented. Additionally, the conceptual design of the bldg resource and its child resources based on the level of details is also presented.

1. INTRODUCTION

The Web Services technology has dramatically affected the development of WebGIS products. A variety of organization publish data and functions via Web Services (Newcomer & Lomow, 2005). Web Services are key components of web applications and represent an important evolution of distributed computing. The main idea of a web service is a collection of smaller programs distributed across the Web, running on different servers, but still communicating with each other and functioning together as a whole (Fu & Sun, 2010). Therefore, Web Services can be published, found and used on the Web (W3Schools, 1999-2016). The development of effective techniques is very important for the management of the modern cities, since it is estimated that in the near future half of the world's population will live in cities (Prandi et al., 2015). Therefore, the proper organization of the city data is considered vital in compliance with the international standard CityGML. CityGML is a common semantic information model for the representation of 3D urban objects that can be shared over different applications. This capability is especially important regarding a cost-effective sustainable maintenance of 3D city models, enabling the same data to be provided to customers from different application fields (Groger, et al., 2012). However, although it is considered as the most appropriate model for the representation of 3D city models, it is quite difficult to manage, modify and retrieve CityGML data based on their semantic, geometric and descriptive features. Taking into account the complex structure of CityGML and the need to retrieve data from distributed sources addressing interoperability problems, the adoption of appropriate Web Service is required. The adoption of Web Services leads to the implementation of WOA (Web - Oriented Architecture) which is a specialization of SOA (Service-Oriented Architecture), utilizing RESTful Web services and lightweight mashups (Thies & Vossen, 2008). Using WOA a full utilization of Web capacity was achieved and hence, the development of reliable, flexible application was facilitated in an easiest and economical way (Kralidis, 2007). The Open Geospatial Consortium (OGC) developed and implemented several Geospatial Web services among which the Web Map Service (WMS), the Web Feature Service (WFS), the Web Coverage Service (WCS) and the Catalogue Service for the Web (CSW). In the context of 3D there are two 3D portrayal services: Web 3D Service (W3DS) (Quadt & Kolbe 2005; Schilling & Kolbe 2010) and Web View Service (WVS) (Benjarmin, 2010). Nevertheless, these services were developed in terms of data visualization mainly based on geometric features. On the other hand, taking into consideration the semantic structure of CityGML and the need for retrieving CityGML data based on their semantics characteristics, the aforementioned OGC Services are not considered important for the purpose of this paper.
The aim of this paper is to provide a new approach for retrieving CityGML data based on their semantic characteristics. To this purpose, the SOAP and REST Web Services are further studied and compared. Thereafter, several principles and guideline are addressed with regard to the new approach on the automatic retrieval of CityGML Data and finally, the conceptual design of the resource and child resources regarding the building module of CityGML is presented.
This paper is structured as follows. Section 2 describes the available Web services, the related work and the reasoning of the REST-style architecture approach. In Section 3 several principles and guidelines of the aforementioned approach are presented and the "CityModels" resource is introduced. Section 4 describes the conceptual design of bldg resource and its respective child resources based on the LoD of the data. Moreover, several case studies of semantics requests are presented. Section 5 concludes the findings and discusses suggestions for future research work.

2. WEB TECHNOLOGIES AND RELATED WORK

2.1 Web Services

The interest in Web services has rapidly increased from the very start. The main goal of web services is to exchange information

* Corresponding author

among applications in the standard way (Mumbaikar & Padiya, 2013). Their exploitation provides a new approach in terms of system interoperability. Namely, it overcomes the complexity of the need to convert data and install the appropriate programs, allowing systems to work at a Web Service level (Fu & Sun, 2010). The clients normally do not have any prior knowledge of web services before using them, and therefore, Web Services are actually platform independent and loosely coupled.
There are two types of Web Services based on SOAP and REST principle: SOAP-based Web Services and REST-style Web Services

2.1.1 SOAP Vs REST: REST has gained widespread acceptance across the Web as a simpler alternative to SOAP-based Web services. Key evidence of this shift in interface design is the adoption of REST by mainstream Web 2.0 service providers-including Yahoo, Google, and Facebook -who have deprecated or passed on SOAP-based interfaces in favor of an easier-to-use, resource-oriented model to expose their services (Rodriguez, 2008). It should be noted that in 2002 Amazon aware of the "REST versus SOAP" debate provides both SOAP and REST interface to its Web Services. As a result, in 2004, 80 percent of the calls to Amazon's Web Services were REST-based (Greenfield and Dornan, 2004). Additionally, the REST language is based on the use of nouns (resources) and verbs (HTTP methods) and hence, they do not require message format like XML envelope which is required in SOAP messages (Mumbaikar & Padiya, 2013). In many cases, the simplicity and efficiency of using REST outweighs the rigorous discipline of SOAP and the complexity in introducing SOAP-based Web Services (Fu & Sun, 2010). Additionally, Mulligan et al. (2009) presented a comparison of SOAP and REST implementations of a service-based interaction independence middleware framework. The results of their tests have shown that the REST implementation of the data transmission component proved to be more efficient in terms of both the network bandwidth and the round-trip latency incurred during these requests. Accordingly, Mumbaikar & Padiya (2013) concluded that SOAP based web services produce considerable network traffic, whereas the RESTful web services are lightweight, easy and Self-descriptive with higher flexibility and lower overhead. Fu & Sun (2010) compared SOAP and REST and referred that the use of REST instead of SOAP brings several advantages to producers, users and managers respectively. More specifically, for producers the cost of creating, hosting and supporting services is lowered. For users the learning curve is reduced and hence, the time and money needed to build GIS applications is also reduced. Finally, for manager the highly desirable architecture properties such as scalability, performance, reliability, and extensibility are provided. However, Kumari (2015) comparing the two protocol concluded that SOAP is preferable for financial, banking, telecommunication services, and REST for Social interaction, Web chat, and mobile services. As a result, SOAP and REST are two different approaches, with different architectural styles, providing several advantages and disadvantages when compared. So, the architectural decision mostly depends on the specific application. It should be noted that SOAP Web Services are robust and comprehensive but complicated. Whilst, REST Web Services are simple and efficient, but may not have all the capabilities of SOAP services. In this paper, after the two protocols are evaluated in relation to the complexity of the CityGML structure, the REST-based architecture style is implemented.

2.1.2 Principles of RESTful Web Services: The evolution of the Web 2.0 phenomenon has led to the increased adoption of the RESTful services paradigm (Lathem et al., 2007). RESTful Web services work on the Web, taking full advantage and making correct use of the HTTP protocol (Webber, 2010). The implementation of the RESTful Web Services must follow the ROA (Resource-Oriented Architecture). Hence, everything that a service provides has to be a resource (Mohedano, Troyano, 2010). The main design constrains of the REST architecture style can be summarized as follows (Webber, 2010):

- Addressability**:** all resources that are published by a Web Service should be given a unique and stable identifier, a URI (Nielsen, 1999). The relationship between URIs and resources is many-to-one. A URI identifies only one resource, but a resource can have more than one URIs (Webber, 2010).

- Uniform Interface: all resources are managed via a uniform interface. In HTTP, the actual Web protocol, the uniform interface comprises the methods (HTTP GET, HTTP POST, HTTP PUT and HTTP DELETE) that can be applied to all Web resource identifier.

- Statelessness: Every HTTP request happens in complete isolation (Mohedano Troyano, 2010). Consequently, REST makes the system really scalable since servers do not keep any information from clients.

-Self-Describing Messages: Services interact by exchanging request and response messages, which contain both the representations of resources and the corresponding meta-data.

- HATEOAS (Hypermedia as the Engine of Application State): The ability of a service to change the set of links that are given to a client, based on the current state of a resource.

However, the main design constraints of the REST architectural style can be adopted incrementally, leading to the definition of the Richardson maturity model for RESTful Web services (Fowler, 2010). Namely, this model breaks down the principal elements of a REST approach into four levels (Figure 1).

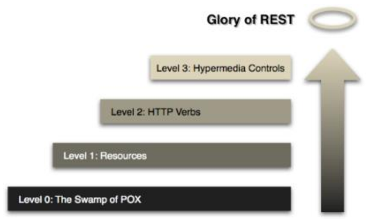


Figure 1. Maturity model for RESTful Web services (Fowler, 2010)

2.2 Related Work

CityGML standard is considered optimal for the presentation of 3D city models. However, because of the complexity of its structure, easy-to–use data retrieval is significant. Several OGC Web Services are currently used to retrieve and portray the CityGML Data, such as W3DS, WVS and WFS. However, the W3DS and WVS are portrayal services, which means that they retrieve a representation of these data (images or scenes) and not real data. Therefore, the most suitable OGC geospatial service is WFS. However, serving CityGML via a WFS presents a number of technical problems relating to the characteristics of the CityGML models and the fact that CityGML schema is much more complex than those usually deployed in WFS. Consequently, the OGC WFS has to be extended so that the retrieval of CityGML data is achieved. Curtis (2008) presented the Snowflake CityGML WFS (Snowflake's GO Publisher) which allows the requests DescribeFeatureType, GetCapabilities and GetFeature, while it supports the following feature of CityGML: Building, CityObjectGroup, GenericCityObject, ReliefFeature and CityFurniture in all LoDs. Within the same research context Kolbe et al. (2015) implemented the OGC WFS in conjunction with 3DcityDB schema, which supports multiscale and rich semantic structure of CityGML (Kolbe et al., 2009), thus developing the 3D City Database WFS. The current WFS version implements the Simple WFS conformance classes and therefore, the 3D city Database WFS handles GetFeatureById queries, not supporting ad-hoc queries or semantic retrieval of available features. The advanced version of the WFS includes more WFS operations and is commercially available (SYSTEMS GmbH Berlin) (Kolbe et al., 2015). Additionally, Zhu, et al. (2016) are investigating for an open source solution to serve CityGML via a WFS with advanced functionality. Such a solution that was tested is GeoServer, combined with its "Application Schema" extension. GeoServer supports the OGC WFS standard and provides full-fledged WFS functionality including discovery, query, locking, transaction and stored query operations (Open Source Geospatial Foundation, 2017). Similarly, a Geoserver approach was implemented by Pispidikis et al. (2016) for the visualization of CityGML data via the WFS standard. Extending WFS to support the retrieval of CityGML data is considered very important. However, the OGC WFS is a geospatial service which means that it is developed with the aim of retrieving, visualizing and modifying data based on Geometry. On the other hand, CityGML's structure is more semantic rather than geometric and therefore the retrieval of the data has to be achieved mainly in compliance with the semantic information. In the context of the aforementioned approach Pispidikis et al. (2016) developed a PHP class which utilize AJAX (Asynchronous JavaScript and XML) techniques with a view to dynamically retrieve CityGML data in JSON format and based on specific semantic characteristics. However, the implementation of this class is not suitable in terms of interoperability. For that, the technology of Web Services are chosen in this paper, and two types of web services (SOAP and REST) are investigated. Thereafter, the REST-style architecture is chosen so that a conceptual model of RESTful Web services is structured and the retrieval of the CityGML data is achieved based on its semantics characteristics.

3. GENERAL OVERVIEW AND PRINCIPLES OF CITYGML RESTFUL WEB SERVICE

The CityGML schema was designed in a way that can be structured according to each application, avoiding the creation of complex files that cannot be checked for their validity (Groger, Kolbe, Nagel, & Hafele, 2012). Consequently, the architecture of the CityGML is shaped via five key components (Figure 2).

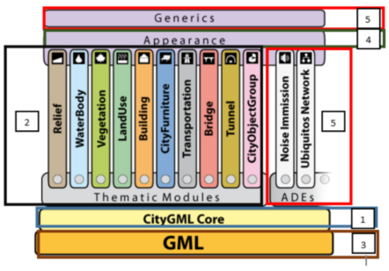


Figure 2. CityGML architecture

The first is the CityGML Core, which defines all the basic classes for CityGML's operation which are inherited by all the CityGML's features (Gröger & Plümer, 2012). The second one, contains the ten thematic modules that define the semantic features of the basic objects of a 3D city model. Worth noting that the implementation of the aforementioned thematic modules is not mandatory but they can be used selectively depending on the application's needs. The third component is the geometric-topological model, which is structured in compliance with the GML3. The fourth component is the appearance module, which defines the observable properties of CityGML's surface objects. Finally, the last component contains the two possible ways that CityGML's scalability is achieved and hence the semantic and descriptive features that are not supported by the current version of CityGML can be added. These ways refer to Generic and ADE (Application Domain Extensions) modules (Groger, Kolbe, Nagel, & Hafele, 2012).

3.1 Principles and Guidelines

There are several guidelines and principles that need to be implemented so that the conceptual design of the CityGML RESTful Web Services can be achieved. Initially, the name of every resource has to be noun and not verb according to the RESTful Web Service guidelines. For instance, a good resource name is the "CityModels" and not the "GetCityModels". The action type of the request is defined by HTTP method. Hence, the RESTful Web service have to be designed in compliance with HTTP specification. So, for the retrieval of the data the HTTP GET method is implemented. Moreover, considering the Richardson maturity model, a good RESTful Web Service has to be in level 3 (see Figure 1). So, every response has to contain links URI to the respectively associate resource.

3.1.1 Geometry implementation: The CityGML data which is retrieved implementing CityGML RESTful Web Service and contains Geometry information is structured in compliance with the geometry object of GeoJSON specification (Gillies et al., 2016) (Figure 3) and not the GML.

```
{
  "type": "object",
  "properties": {
    "type": {"type": "string"},
    "coordinates": {"type": "object"}
  }
}
```

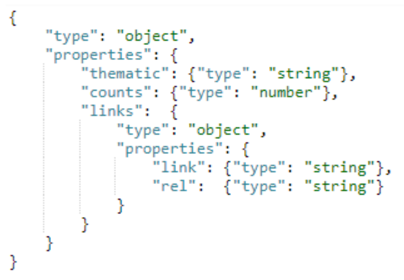
Figure 3. Geometry object based on GeoJSON specification

The "type" member of a geometry object could be Point, MultiPoint, LineString, MultiLineStrings, Polygon, and MultiPolygon. Moreover, the "coordinates" property is composed of one position (in the case of a Point geometry), an array of positions (LineString or MultiPoint geometries), an array of arrays of positions (Polygons, MultiLineStrings), or a multidimensional array of positions (MultiPolygon).

3.1.2 HATEOAS implementation: The implementation of HATEOAS (Hypermedia As the Engine of Application State), an essential part of REST, is considered vital for the CityGML Web Service to be RESTful. As a result, each resource must contain information regarding links to other available resources. Consequently, the available links of a resource of CityGML RESTful Web Service contain links to itself, to all parents' resources and to a child resource. For instance, a wallsurface resource contains the following links: itself, its building resource and available windows and doors resources.

3.2 CityModels Resource

CityGML RESTful Web Services should enable users to have an overview of the available thematic models. Hence, the original resource is called "CityModels". The said resource retrieves the total number of the available thematic models grouped by thematic category model. Moreover, in each group category, the corresponding resource link of the specific main thematic resource (see table 1) will also be retrieved. Additionally, every resource should have link to itself. Hence, the basic schema of the retrieval data from "CityModels" Resource in JSON format is shown in Figure 4.



```
{
    "type": "object",
    "properties": {
        "thematic": {"type": "string"},
        "counts": {"type": "number"},
        "links":  {
            "type": "object",
            "properties": {
                "link": {"type": "string"},
                "rel":  {"type": "string"}
            }
        }
    }
}
```

Figure 4. CityModels Resource Schema

3.2.1 Filter parameters for "CityModels" resource: The "CityModels" resource retrieves information regarding the available CityGML thematic modules. Therefore, the definition of some parameters is considered necessary so that the filtering of the retrieval result can be achieved. As a result, a new filter parameter called "Thematics" is defined. The values of this filter are based on the respective namespace prefix of the thematic modules of CityGML specification. It should be noted that multi thematic values can be used simultaneously by separating them using comma punctuation. For instance, the following request retrieves only the available buildings and bridges of CityGML.

../CityModels?Thematics= bldg,brid

Moreover, the filter BBox is defined. The value of the specified filter is a geometry rectangle in a specific reference system which limits the retrieval result according to the rectangle. The aforementioned filters can be implemented simultaneously.

3.3 Main Thematic Resources

Taking into consideration the five components of the CityGML's architecture (see Figure 2), only the second one (the ten thematic modules) defines the semantic features of CityGML. Therefore, these thematic modules should be the main resources of the CityGML RESTful Web Service. The names of the available main resources of the CityGML RESTful Web service are based on the namespace prefix of CityGML specification and they are shown in table 1.

Resource Name	URI	CityGML Modules
bldg	../ bldg	Buildings
wtr	../ wtr	Waterbodies
dems	../ dem	reliefs
veg	../ veg	Vegetation
luse	../ luse	LandUses
frn	../ frn	CityFurniture
tran	../ tran	Transportations
brids	../ brid	Bridges
tun	../ tun	Tunnels
grp	../ grp	CityObjectGroups

Table 1. Name of the main resources

The response of a request implementing the main thematic resources is mainly a list of the available thematic modules respectively. Each thematic module of list contains general information in compliance with CityGML specification.

3.3.1 General filters: The common attributes that are supported by all thematic modules of CityGML are function, usage and class. As a result, these attributes are use d as filter parameters of Main Thematic resources and their values are defined by CityGML specification. Additionally, BBox filter parameters is vital to be defined so that the retrieval data to be filtered based on spatial queries. Moreover, CityGML supports multi-scale modelling with five different LoDs (levels of detail). In a CityGML data set, the same object can be represented at different levels of detail simultaneously, thereby allowing the analysis and visualization of the latter to varying degrees of spatial analysis (Groger, Kolbe, Nagel, & Hafele, 2012). However, LoD is considered vital not only in the geometric determination of the level of detail but also in the semantic. By increasing the LoD, the semantic richness of CityGML increases respectively. Hence, LoD is implemented as filter parameter by all main thematic resources of CityGML RESTful Web Service so that the semantic information results are limited based on level of detail. Consequently, the values of the LoD filter parameter are 0 to 4. The set of the available general filter of the main thematic resource is shown in table 2. The available filters can be implemented simultaneously.

Filter	URI (Example for bldg resource)
Function	../bldg?function=3020
Usage	../ bldg?usage=1010
Class	../ bldg?class =1000
BBox	../ bldg?BBox= 334433.0,4455667.0,445677.0,5566556.0
Lod	/bldg?lod=2

Table 2. General filters of Main Thematic resources

4. CONCEPTUAL DESIGN OF BLDG RESOURCE

The building module is considered as one of the most detailed thematic concepts of CityGML, allowing the representation of thematic and spatial parameters of buildings and building sections at different levels of detail (Groger, Kolbe, Nagel, & Hafele, 2012). The transition from one level to another imposes and allows different semantic and geometric details both on the outside and inside.

4.1 Bldg Resource

The main thematic resource regarding building module of CityGML is the bldg resource. It is quite important that the said resource of CityGML RESTful Web Service enables the users to retrieve any of the available semantic features of the respective

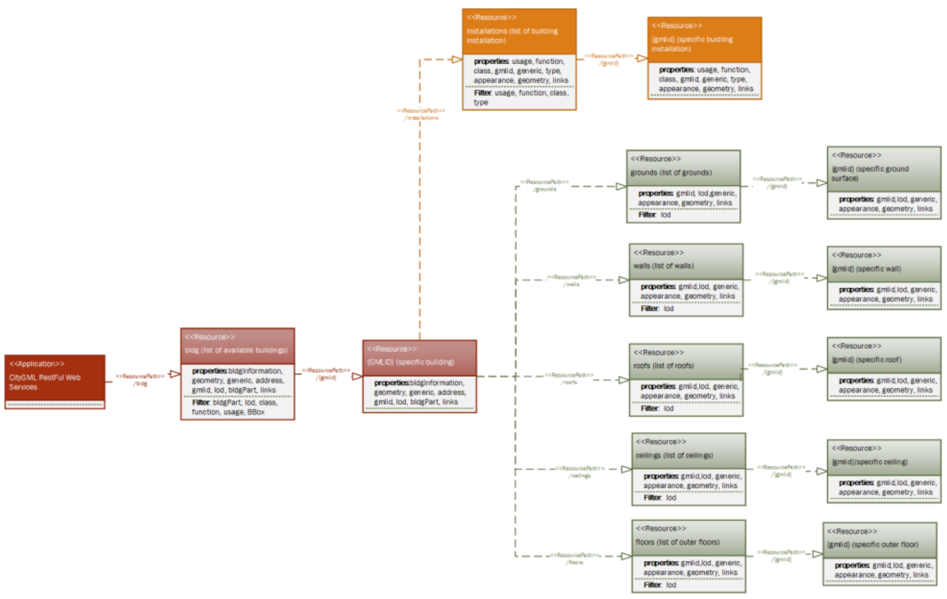

<<Application>>
CityGML RestFul Web Services
<<ResourcePath>> /bldg
<<Resource>>
bldg (list of available buildings)
properties: bldgInformation, geometry, generic, address, gmlid, lod, bldgPart, links
Filter: bldgPart, lod, class, function, usage, BBox
<<ResourcePath>> /{gmlid}
<<Resource>>
{GMLID} (specific building)
properties: bldgInformation, geometry, generic, address, gmlid, lod, bldgPart, links
<<ResourcePath>> /installations
<<Resource>>
installations (list of building installation)
properties: usage, function, class, gmlid, generic, type, appearance, geometry, links
Filter: usage, function, class, type
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific buidling installation)
properties: usage, function, class, gmlid, generic, type, appearance, geometry, links
<<ResourcePath>> /grounds
<<Resource>>
grounds (list of grounds)
properties: gmlid, lod, generic, appearance, geometry, links
Filter: lod
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific ground surface)
properties: gmlid, lod, generic, appearance, geometry, links
<<ResourcePath>> /walls
<<Resource>>
walls (list of walls)
properties: gmlid, lod, generic, appearance, geometry, links
Filter: lod
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific wall)
properties: gmlid, lod, generic, appearance, geometry, links
<<ResourcePath>> /roofs
<<Resource>>
roofs (list of roofs)
properties: gmlid, lod, generic, appearance, geometry, links
Filter: lod
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific roof)
properties: gmlid, lod, generic, appearance, geometry, links
<<ResourcePath>> /ceilings
<<Resource>>
ceilings (list of ceilings)
properties: gmlid, lod, generic, appearance, geometry, links
Filter: lod
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid}(specific ceiling)
properties: gmlid, lod, generic, appearance, geometry, links
<<ResourcePath>> /floors
<<Resource>>
floors (list of outer floors)
properties: gmlid, lod, generic, appearance, geometry, links
Filter: lod
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific outer floor)
properties: gmlid, lod, generic, appearance, geometry, links

Figure 5. LoD2 conceptual design

building. Thus, the bldg resource retrieves a list of the available buildings and buildingParts respectively. This list can be limited implementing several filters. Except for the general filters (see 3.3.1) which are used in all main thematic resources, the bldgPart filter must be implemented. The value of this filter is Boolean and provides information about whether the building is BuildingPart or not. The retrieval of a specific building can be achieved implementing the following resource:

../bldg/{gmlid}

The gmlid in the brackets is the unique id for each building according to CityGML. The available information of this resource is shown in table 3.

Information	Type	Description
lod	Number	LoD value
bldgPart	Boolean	True or False
bldgInfomation	Object	List of key value pairs based on building module
geometry	Object	Geometry object based on GeoJSON specification
generic	Object	Ad hoc list of key value pairs based on generic module
address	Object	List of key value pairs based on xAL specification
links	Object	List of key value pairs regarding links to the parent and child resources
gmlid	String	Gmlid value
terrain	String	

Table 3. Available information of bldg resource

However, the building module is enriched by semantics characteristics from LoD2. Hence, the child resources of bldg resource are based on semantic data of building module from LoD2 to LoD4.

4.1.1 LoD2 bldg sub-resources: The supported semantic characteristics of the LoD2 building are the exterior boundary surface (WallSurface, RoofSurface, GroundSurface, OuterCeilingSurface and OuterFloorSurface) and the exterior building installation. As a result, these semantic features are the LoD2 child resources of the bldg resource. The URI resources regarding boundary surfaces are walls, roofs, grounds, ceilings and floors respectively. These resources retrieve a list of the corresponding thematic surfaces and this list can be filtered implemented lod filter parameter. Additionally, the exterior building installation resource is called “installations”. This resource retrieves a list of the exterior building installations and can be filtered using several filters such as usage, function, class and type. It should be noted that the ”installations” resource refers both for interior and exterior building installations. The separation of the latter is achieved via the “type” property. Thereby, the defined values of this property are interior or exterior respectively. However, the interior building installations are semantic features available in LoD4. Hence, the “installations” resource defined as a child sub-resource regarding LoD4 as well. Additionally, the retrieval of a specific resource can be achieved using the corresponding gmlid. An instance of a specific wall request is the following:

../bldg/ {gmlid}/walls/{gmlid}

The available information of each semantic surface of LoD2 bldg sub-resources is shown in table 4. Moreover, the conceptual design of the bldg resource with available properties and filters according to LoD2 is shown in Figure 5.

Information	Type	Resource	Description
lod	Number	Installations, Exterior boundaries*	LoD value
appearance	Object	Installations, Exterior boundaries*	List of key value pairs based on appearance module
geometry	Object	Installations, Exterior boundaries*	Geometry object based on GeoJSON specification
generic	Object	Installations, Exterior boundaries*	Ad hoc list of key value pairs based on generic module

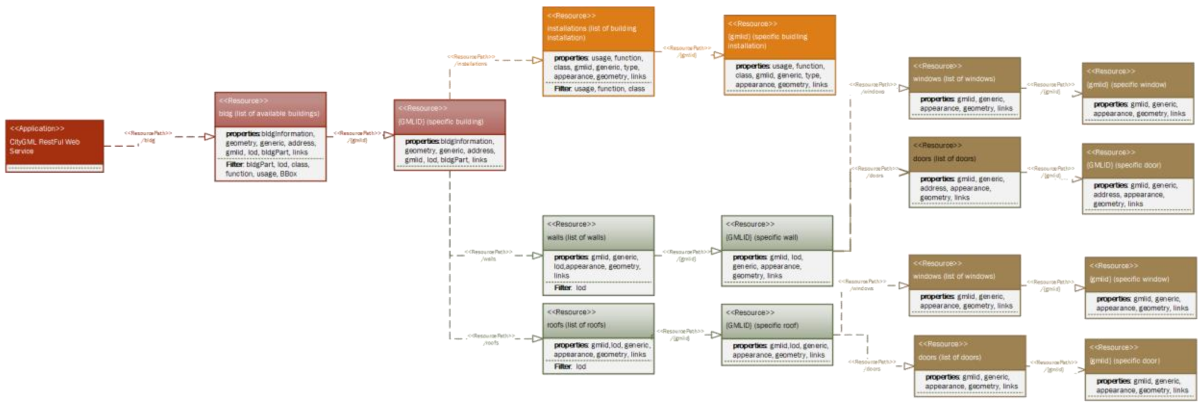

<<Application>>
CityGML RestFul Web Service
<<ResourcePath>> /bldg
<<Resource>>
bldg (list of available buildings)
properties: bldgInformation, geometry, generic, address, gmlid, lod, bldgPart, links
Filter: bldgPart, lod, class, function, usage, BBox
<<ResourcePath>> /{gmlid}
<<Resource>>
{GMLID} (specific building)
properties: bldgInformation, geometry, generic, address, gmlid, lod, bldgPart, links
<<ResourcePath>> /installations
<<Resource>>
installations (list of building installation)
properties: usage, function, class, gmlid, generic, type, appearance, geometry, links
Filter: usage, function, class
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific building installation)
properties: usage, function, class, gmlid, generic, type, appearance, geometry, links
<<ResourcePath>> /walls
<<Resource>>
walls (list of walls)
properties: gmlid, generic, lod,appearance, geometry, links
Filter: lod
<<ResourcePath>> /{gmlid}
<<Resource>>
{GMLID} (specific wall)
properties: gmlid, lod, generic, appearance, geometry, links
<<ResourcePath>> /roofs
<<Resource>>
roofs (list of roofs)
properties: gmlid,lod, generic, appearance, geometry, links
Filter: lod
<<ResourcePath>> /{gmlid}
<<Resource>>
{GMLID} (specific roof)
properties: gmlid,lod, generic, appearance, geometry, links
<<ResourcePath>> /windows
<<Resource>>
windows (list of windows)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific window)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>> /doors
<<Resource>>
doors (list of doors)
properties: gmlid, generic, address, appearance, geometry, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{GMLID} (specific door)
properties: gmlid, generic, address, appearance, geometry, links
<<ResourcePath>> /windows
<<Resource>>
windows (list of windows)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific window)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>> /doors
<<Resource>>
doors (list of doors)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific door)
properties: gmlid, generic, appearance, geometry, links

Figure 6. LoD3 conceptual design

links	Object	Installations, Exterior boundaries*	List of key value pairs regarding links to the parent and child resources
gmlid	String	Installations, Exterior boundaries*	gmlid value
usage	Number	installations	LoD value
function	String	installations	function value
class	String	installations	class value
type	String	installations	exterior or interior

Exterior boundaries*: walls, roofs, grounds, ceilings and floors

Table 4. Available information of LoD2 bldg sub-resources

4.1.2 LoD3 bldg sub-resources: The additional semantic features of the LoD3 building module are the opening features (windows and doors). Consequently, the respective resources of the aforesaid features are considered vital to be defined. Hence, the URI of these resources are windows and doors. The retrieval of specific data regarding the aforementioned resources is achieved implementing the corresponding gmlid as a sub-resource.

Information	Type	Description
generic	Object	Ad hoc list of key value pairs based on generic module
Appearance	Object	List of key value pairs based on appearance module
Geometry	Object	Geometry object based on GeoJSON specification
Links	Object	List of key value pairs regarding links to the parent and child resources
address*	Object	List of key value pairs based on xAL specification
gmlid	String	gmlid string

Address*: available for 'doors' sub-resource

Table 5. Available information of windows and doors sub-resources

Additionally, each specific "doors" sub-resource should contain information regarding the address and hence, the object address is defined.

The conceptual design of the bldg child resources regarding LoD 3 is shown in figure 6 while the respective retrieval information from the windows and doors resources of LoD3 are described in table 5.

4.1.3 LoD4 bldg sub-resources: Except for installations resource which retrieves interior building installations of building, there is the "rooms" child resource regarding LoD4 as well. Therefore, this resource retrieves the list of the available rooms of a building. Similarly, the retrieval of specific room is achieved implementing the respective gmlid. Moreover, the available information of each room is class, usage, function, gmlid, links and generic. The filtering of this resource is implemented using the class, function, usage and BBox filter parameters respectively. Thereafter, each room provides several links for child resources such as furniture, installations, walls, floors and ceiling. The first one retrieves a list of furniture that are located in a specific room. The accessible information of this resource is class, usage, function, gmlid, generic, appearance, geometry and links. Additionally, the available filter parameters of the furniture resource are class, usage and function. In this context, the rest child resources (installations, walls, floors and ceilings) retrieve a list of the respective available semantic features. The accessible retrieval information and the corresponding filters are shown in figure 7. Generally, the retrieval of a specific semantic feature is achieved using the gmlid. It should be noted that in the LoD4 there are two sub resources with the same name but different URIs. The name of these resources is named "installations". The first one is child resource of bldg resource and retrieve a list of interior installations in a specific building, while the second one is the child resource of the "rooms" resource and retrieve the respective installations that are located in a specific room. Similar to the LoD3, the interior boundary resources (walls, floors and ceilings) provide the windows and doors child resources. The aforesaid resources have similar properties and filters like LoD3 opening resources (see table 5). The conceptual design of LoD4 bldg sub resources are shown in Figure 7.

4.2 Case Studies of Semantic Requests

In this section, several requests are presented using the conceptual design of the CityGML RESTful Web Service regarding the bldg resource. Initially, it should be noted that the code list values of function, usage and class regarding the buildings, interior/exterior installations, rooms and furniture is specified in the XML file CityGML_ExternalCodeLists.xml, according to the dictionary concept of GML 3.

4.2.1. Basic requests:

- Overview of the available buildings in LoD2:

```
../CityModels?Thematics= bldg&lod=2
```

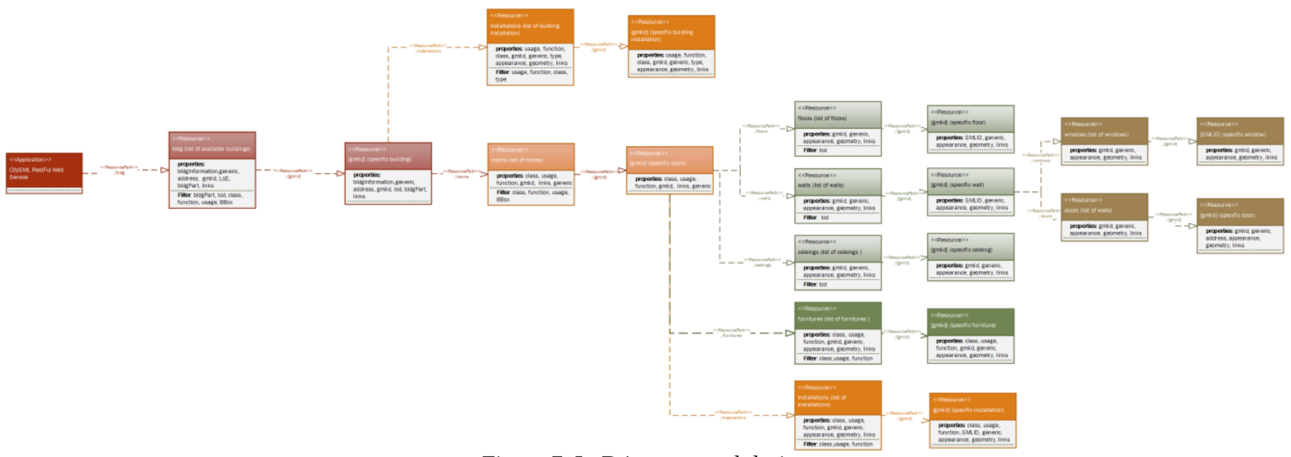

<<Application>> CityGML RestFul Web Service
<<ResourcePath>> /bldg
<<Resource>> bldg (list of available buildings)
properties: bldgInformation,generic, address, gmlid, LoD, bldgPart, links
Filter: bldgPart, lod, class, function, usage, BBox
<<ResourcePath>> /{gmlid}
<<Resource>> {gmlid} (specific building)
properties: bldgInformation,generic, address, gmlid, lod, bldgPart, links
<<ResourcePath>> /installations
<<Resource>> installations (list of building installation)
properties: usage, function, class, gmlid, generic, type, appearance, geometry, links
Filter: usage, function, class, type
<<Resource>> {gmlid} (specific building installation)
properties: usage, function, class, gmlid, generic, type, appearance, geometry, links
<<ResourcePath>> /rooms
<<Resource>> rooms (list of rooms)
properties: class, usage, function, gmlid, links, generic
Filter: class, function, usage, BBox
<<Resource>> {gmlid} (specific room)
properties: class, usage, function, gmlid, links, generic
<<ResourcePath>> /floors
<<Resource>> floors (list of floors)
properties: gmlid, generic, appearance, geometry, links
Filter: lod
<<Resource>> {gmlid} (specific floor)
properties: GMLID, generic, appearance, geometry, links
<<ResourcePath>> /walls
<<Resource>> walls (list of walls)
properties: gmlid, generic, appearance, geometry, links
Filter: lod
<<Resource>> {gmlid} (specific wall)
properties: GMLID, generic, appearance, geometry, links
<<ResourcePath>> /windows
<<Resource>> windows (list of windows)
properties: gmlid, generic, appearance, geometry, links
<<Resource>> {GMLID} (specific window)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>> /doors
<<Resource>> doors (list of walls)
properties: gmlid, generic, appearance, geometry, links
<<Resource>> {gmlid} (specific door)
properties: gmlid, generic, address, appearance, geometry, links
<<ResourcePath>> /celeings
<<Resource>> celeings (list of celeings)
properties: gmlid, generic, appearance, geometry, links
Filter: lod
<<Resource>> {gmlid} (specific celeing)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>> /furnitures
<<Resource>> furnitures (list of furnitures)
properties: class, usage, function, gmlid, generic, appearance, geometry, links
Filter: class,usage, function
<<Resource>> {gmlid} (specific furniture)
properties: class, usage, function, gmlid, generic, appearance, geometry, links
<<Resource>> installations (list of installations)
properties: class, usage, function, gmlid, generic, appearance, geometry, links
Filter: class,usage, function
<<Resource>> {gmlid} (specific installation)
properties: class, usage, function, GMLID, generic, appearance, geometry, links

Figure 7. LoD4 conceptual design

- The buildings with function hostel and residential in specific boundary area (e.g. 334433.0,4455667.0,445677.0,5566556.0)

../bldg?function=1000,1020&BBox=334433.0,4455667.0,445677.0,5566556.0

- The walls of building 2

../bldg/2/walls

- The windows of the wall with gmlid 2 for building 1

../bldg/1/walls/2/windows

- The Light switches in building 2

../bldg/2/installations?function=3020

- The radiators of room 3 for building 2

../bldg/2/rooms/3/installations?usage=1010

- The living room in building 3

../bldg/3/rooms?function=1000

- The windows of room 2 for building 3

../bldg/3/rooms/2/windows

- The furniture of room 3 for building 4

../bldg/4/rooms/3/furniture

- The lamps of room 2 for building 3

../bldg/3/rooms/2/installations?function=3010

4.2.2. Advanced requests: Initially, each HTTP request should happen in complete isolation (stateless interaction). As a result, when the retrieval information is complex and need more than one request to be used then these requests have to be implemented sequentially. Hence, the result of each request can be used as input value for the next request. However, taking into consideration that the CityGML RESTful Web service is designed in compliance with HATEOAS constrain then the URI of every next request can be retrieved from the "links" object of the current request.

- The doors of the toilet for building 2

.../bldg/2/rooms?function=1050 1st request

Second request using the retrieval gmlid (e.g. 2):

.../bldg/2/rooms/2/doors 2nd request

4.2.3. Requests using simple JavaScript code:

- The number of burned out lamps in the living room for the building with gmld2. Noted that the information about whether the lamps are burned out or not is specified as a generic attribute with the following key value pair: burned: Boolean.

The implementation of the first request retrieves the gmlid of the living room:

.../bldg/2/rooms?function=1000 1st request

The second request is implemented utilizing the retrieval gmlid (e,g, 3) in conjunction with the respective filter regarding the installation function type:

.../bldg/2/rooms/3/installations?function=3010 2nd request

Thereafter, the retrieval result is implemented as JSON input in JavaScript code:

```
var Count=0;
responce.forEach(function(installations){
If(installations.generic.burned==true) {
  Count++;  }})
Count;                              Result
```

5. CONCLUSION

In this paper, a new approach regarding the automatic retrieval of CityGML data, based on their semantic characteristic is presented. This approach is the CityGML RESTful Web service. To this purpose, the two types of web services based on SOAP and REST principle are thoroughly studied and compared. As a result, the REST-style Web Service is chosen. The aforesaid choice is considered more appropriate in comparison to the OGC WFS, since the OGC WFS is a geospatial service developed with the aim of retrieving, visualizing and modifying data based on their geometry characteristics. Whereas, CityGML RESTful Web service retrieves CityGML data according to semantic features. Thereafter, several principles and guidelines are addressed and the CityModel resource is introduced. Finally, taking into account that the building module is one of the most detailed thematic concepts of CityGML, the conceptual design of bldg resource and the respective child resources regarding LoD2 to LoD4 are described and presented. However, only the first of the ten main thematic resources is conceptually designed, while the conceptual design of the rest main resources for all LoDs needs to be further examined. It should be noted that this approach focuses on the retrieval of the CityGML data and not on the management of data (update and edit). Finally, when the conceptual design is accomplished the logical and physical design of this approach may be implemented and Cross-Domain issues to be examined.

ACKNOWLEDGEMENTS

The Scholarship from the Onassis Foundation for this research is gratefully acknowledged.

REFERENCES

Benjarmin, H. (2010). Web View Service Discussion Paper. *ver 0.3.0*. OpenGIS Discussion Paper.

Curtis, E. (2008). Serving CityGML via web feature services in the OGC web services-phase 4 testbeds. Journal of Advances in 3D geoinformation systems, 331-340

Fowler, M. (2010). Richardson Maturity Model: steps toward the glory of REST. Online at http://martinfowler. com/articles/richardsonMaturityModel. Html.

Fu, P., & Sun, J. (2010). *Web GIS: principles and applications.* USA: Esri Press.

Gillies, S., Butler, H., Daly, M., Doyle, A., & Schaub, T. (2016). The GeoJSON Format. coordinates, 102, 0-5.

Greenfield, D., & Dornan, A. (2004). Amazon: Web Site to Web Services. Network Magazine, 19(10), 58-60.

Groger, G., Kolbe, T., Nagel, C., & Hafele, K.-H. (2012). OGC City Geography Markup Language (CItyGML) Encoding Standard. Retrieved from Open Geospatial Consortium Inc (2017): www.opengis.net/spec/citygml/2.0

Gröger, G., & Plümer, L. (2012). CityGML – Interoperable semantic 3D city models. ISPRS Journal of Photogrammetry and Remote Sensing (71), pp. 12–33.

Kolbe, T H., Zhihang Y., Nagel, C., Felix Kunde, Philipp Willkomm, György Hudra, Arda Müftüoglu & Javier Herreruela. (2015). 3D-City Database for CityGML. Institute for Geodesy and Geoinformation Science, Technische Universität Berlin, 3.

Kolbe, T. H., König, G., Nagel, C., & Stadler, A. (2009). 3D-Geo-Database for CityGML. Institute for Geodesy and Geoinformation Science, Technische Universität Berlin, 2(1).

Kralidis, T. A. (2007). Geospatial web services: the evolution of geospatial data infrastructure. Springer London

Kumari, V. (2015). Web Services Protocol: SOAP vs REST. International Journal of Advanced Research in Computer Engineering & Technology (IJARCET), 4(5), 2467-2469.

Lathem, J., Gomadam, K., & Sheth, A. P. (2007). Sa-rest and (s) mashups: Adding semantics to restful services. In Semantic Computing, 2007. ICSC 2007. International Conference IEEE. pp. 469-476.

Mohedano Troyano, N. (2010). The design of a RESTful web-service (Master's thesis, Universitat Politècnica de Catalunya).

Mueller, J. (2013). Understanding SOAP and REST Basics And Differences. Retrieved from SMARTBEAR (2016): http://blog.smartbear.com/apis/understanding-soap-and-rest-basics/

Mulligan, G., & Gračanin, D. (2009, December). A comparison of SOAP and REST implementations of a service-based interaction independence middleware framework. In Winter Simulation Conference pp. 1423-1432

Mumbaikar, S., & Padiya, P. (2013). Web services based on soap and rest principles. International Journal of Scientific and Research Publications, 3(5), 1-4.

Newcomer, E., & Lomow, G. (2005). Understanding SOA with Web services. Addison-Wesley

Nielsen, J. (1999). User interface directions for the web. Communications of the ACM, 42(1), 65-72.

Open Source Geospatial Foundation. (2017). GeoServer User Manual, 2.12.x. Retrieved from GeoServer (2017): http://docs.geoserver.org/latest/en/user/

Pispidikis, I., & Dimopoulou, E. (2016). Development of a 3D WebGIS system for retrieving and visualizing CityGML data based on their geometric and semantic characteristics by using free and open source technology. ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences, 4(2).

Potociar, M. (2011). When to use SOAP and when REST. In International Conference on the Modern Art of Software, 5 th Annual.

Prandi, F., Devigili, F., Soave, M., Di Staso, U., & De Amicis, R. (2015). 3D web visualization of huge CityGML models. The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, 40(3), 601.

Quadt, U., & Kolbe, T. (2005, Febrouary 2). Web 3D Service.

Richardson, L., & Ruby, S. (2007). RESTful web services. USA: O'Reilly Media, Inc

Rodriguez, A. (2008). Restful web services: The basics. IBM developerWorks.

Schilling, A., & Kolbe, T. H. (2010). Draft for Candidate OpenGIS Web 3D Service Interface Standard. *ver. 0.4.0*. OpenGIS Discussion Paper

Thies, G., & Vossen, G. (2008). Web-oriented architectures: On the impact of Web 2.0 on service-oriented architectures. IEEE Asian-Pacific Services Computing Conference, (pp. 1075-82).

Webber, J., Parastatidis, S., & Robinson, I. (2010). REST in practice: Hypermedia and systems architecture. O'Reilly Media, Inc.".

Zhu, W., Simons, A., Wursthorn, S., & Nichersu, A. (2016). Integration of CityGML and Air Quality Spatio-Temporal Data Series via OGC SOS. In Conference proceedings Geospatial Sensor Web Conference.