National Technical University of Athens

School of Rural and Surveying Engineering

Department of Topography

OPTIMIZATION OF AUTOMATED RETRIEVAL OF SEMANTIC 3D CITY DATA

DOCTORAL DISSERTATION

for the title of Doctor of Philosophy in Engineering submitted to the School of Rural & Surveying Engineering, National Technical University of Athens

IOANNIS S. PISPIDIKIS
Bachelor in Military Operational Art and Science H.A.A
Diploma of Rural and Surveying Engineering N.T.U.A
M.Sc in Geoinformatics N.T.U.A
Officer in H.M.G.S

SUPERVISOR:

EFI DIMOPOULOU

Professor N.T.U.A.

ATHENS, November 2020

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

Σχολή Αγρονόμων Τοπογράφων Μηχανικών

Τομέας Τοπογραφίας

ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ ΤΗΣ ΑΥΤΟΜΑΤΗΣ ΑΝΑΚΤΗΣΗΣ ΣΗΜΑΣΙΟΛΟΓΙΚΩΝ 3D ΔΕΔΟΜΕΝΩΝ ΠΟΛΗΣ

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

για τον Επιστημονικό Τίτλο του Διδάκτορα Μηχανικού υποβληθείσα στη Σχολή Αγρονόμων και Τοπογράφων Μηχανικών του Εθνικού Μετσόβιου Πολυτεχνείου

ΙΩΑΝΝΗΣ Σ. ΠΙΣΠΙΔΙΚΗΣ

Διπλωματούχος Αξιωματικός στις Στρατιωτικές Επιστήμες Σ.Σ.Ε.

Διπλωματούχος Αγρονόμος και Τοπογράφος Μηχανικός Ε.Μ.Π.

M.Sc στη Γεωπληροφορική Ε.Μ.Π.

Αξιωματικός της Γ.Υ.Σ

ΕΠΙΒΛΕΠΟΥΣΑ:

ΕΦΗ ΔΗΜΟΠΟΥΛΟΥ

Καθηγήτρια Ε.Μ.Π.

ΑΘΗΝΑ, Νοέμβριος 2020

National Technical University of Athens

School of Rural and Surveying Engineering

Department of Topography

OPTIMIZATION OF AUTOMATED RETRIEVAL OF SEMANTIC 3D CITY DATA

DOCTORAL DISSERTATION

for the title of Doctor of Philosophy in Engineering submitted to the School of Rural and Surveying Engineering, National Technical University of Athens

IOANNIS S. PISPIDIKIS

Bachelor in Military Operational Art and Science H.A.A

Diploma of Rural and Surveying Engineering N.T.U.A

M.Sc in Geoinformatics N.T.U.A

Officer in H.M.G.S

ADVISORY COMMITTEE

1. E. DIMOPOULOU, Prof. N.T.U.A (supervisor)
2. I. PSARRAS, Prof. N.T.U.A
3. V. VESCOUKIS, As. Prof. N.T.U.A.

EXAMINIATION COMMITTEE

1. E. DIMOPOULOU, Prof. N.T.U.A (supervisor)
2. I. PSARRAS, Prof. N.T.U.A
3. V. VESCOUKIS, As. Prof. N.T.U.A.
4. D. ASKOUNIS, Prof. N.T.U.A
5. N. DOULAMIS, As. Prof. N.T.U.A.
6. H. DOUKAS, As. Prof. N.T.U.A.
7. M. KOKLA, Lect. N.T.U.A.

ATHENS, November 2020

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

Σχολή Αγρονόμων Τοπογράφων Μηχανικών

Τομέας Τοπογραφίας

ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ ΤΗΣ ΑΥΤΟΜΑΤΗΣ ΑΝΑΚΤΗΣΗΣ ΣΗΜΑΣΙΟΛΟΓΙΚΩΝ 3D ΔΕΔΟΜΕΝΩΝ ΠΟΛΗΣ

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

για τον Επιστημονικό Τίτλο του Διδάκτορα Μηχανικού υποβληθείσα στη Σχολή Αγρονόμων και Τοπογράφων Μηχανικών του Εθνικού Μετσόβιου Πολυτεχνείου

ΙΩΑΝΝΗΣ Σ. ΠΙΣΠΙΔΙΚΗΣ

Διπλωματούχος Αξιωματικός στις Στρατιωτικές Επιστήμες Σ.Σ.Ε.

Διπλωματούχος Αγρονόμος Τοπογράφος Μηχανικός Ε.Μ.Π.

M.Sc στη Γεωπληροφορική Ε.Μ.Π

Αξιωματικός της Γ.Υ.Σ

ΣΥΜΒΟΥΛΕΥΤΙΚΗ ΕΠΙΤΡΟΠΗ

1. Ε. ΔΗΜΟΠΟΥΛΟΥ, Καθ. Ε.Μ.Π. (επιβλέπουσα)
2. Ι. ΨΑΡΡΑΣ, Καθ. Ε.Μ.Π.
3. Β. ΒΕΣΚΟΥΚΗΣ, Αν. Καθ. Ε.Μ.Π.

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ

1. Ε. ΔΗΜΟΠΟΥΛΟΥ, Καθ. Ε.Μ.Π. (επιβλέπουσα)
2. Ι. ΨΑΡΡΑΣ, Καθ. Ε.Μ.Π.
3. Β. ΒΕΣΚΟΥΚΗΣ, Αν. Καθ. Ε.Μ.Π
4. Δ. ΑΣΚΟΥΝΗΣ, Καθ. Ε.Μ.Π.
5. Ν. ΔΟΥΛΑΜΗΣ, Αν. Καθ. Ε.Μ.Π.
6. Χ. ΔΟΥΚΑΣ, Αν. Καθ. Ε.Μ.Π.
7. Μ. ΚΟΚΛΑ, Λεκτ. Ε.Μ.Π

ΑΘΗΝΑ, Νοέμβριος 2020

« Η έγκριση της διδακτορικής διατριβής από την Ανώτατη Σχολή Αγρονόμων και Τοπογράφων Μηχανικών του Ε.Μ. Πολυτεχνείου δεν υποδηλώνει αποδοχή των γνωμών του συγγραφέα (Ν. 5343/1932, Άρθρο 202) ».

This research was funded by a scholarship awarded by the Onassis Foundation.

«Το μυαλό δεν είναι ένα δοχείο που πρέπει να γεμίσει,
αλλά μια φωτιά που πρέπει ν' ανάψει».

Πλούταρχος

ΕΥΧΑΡΙΣΤΙΕΣ

Η παρούσα διδακτορική διατριβή διεξήχθη στη γνωστική περιοχή του Κτηματολογίου και της Γεωπληροφορικής, στον τομέα Τοπογραφίας της Σχολής Αγρονόμων και Τοπογράφων Μηχανικών του Εθνικού Μετσοβίου Πολυτεχνίου. Η εκπόνηση της εν λόγω μελέτης δε θα ήταν εφικτή, αν δεν είχα την αμέριστη συμπαράσταση και αρωγή ορισμένων ανθρώπων, τους οποίους αισθάνομαι την ανάγκη να ευχαριστήσω.

Ιδιαίτερες ευχαριστίες θα ήθελα να απευθύνω στη Καθηγήτριά Έφη Δημοπούλου, της Σχολής Αγρονόμων και Τοπογράφων Μηχανικών του ΕΜΠ, για την εμπιστοσύνη που μου έδειξε, καθώς επίσης και για τη συνεχή επιστημονική καθοδήγηση και ηθική στήριξη που μου παρείχε καθ' όλη τη διάρκεια της μακροχρόνιας συνεργασίας μας, η οποία ξεκίνησε από τις προπτυχιακές μου σπουδές. Ακόμη, αξίζει μνεια το γεγονός πως με τις πολύτιμες επισημάνσεις, την αμεσότητα, τα προσεκτικά σχόλια και τις διευρυμένες ακαδημαϊκές γνώσεις της με βοήθησε να επιτύχω ένα άρτιο αποτέλεσμα.

Πολλές ευχαριστίες θα ήθελα να εκφράσω και στα υπόλοιπα μέλη της Τριμελούς Συμβουλευτικής Επιτροπής, και συγκεκριμένα προς τον Καθηγητή Ψαρρά Ιωάννη και τον Αναπληρωτή Καθηγητή Βεσκούκη Βασίλειο, για την καθοδήγησή τους κατά την εκπόνηση της παρούσας διατριβής. Αντίστοιχα, θα ήθελα να ευχαριστήσω και τα λοιπά μέλη της Επταμελούς Εξεταστικής Επιτροπής, τον Καθηγητή Ασκούνη Δημήτριο, τον Αναπληρωτή Καθηγητή Δουλάμη Νικόλαο, τον Αναπληρωτή Καθηγητή Χάρη Δούκα και τη Λέκτορα Μαργαρίτα Κόκλα, για την αποδοχή τους να συμμετέχουν στην κρίση της προκείμενης έρευνας.

Επιπρόσθετα, θα ήθελα να ευχαριστήσω τον Τάσο Λαμπρόπουλο, καθώς και την ερευνητική ομάδα του 3D Campus, Ευτυχία Καλογιάννη, Δημήτριο Κιτσάκη, Κατερίνα Αθανασίου και Εύα Τσιλιάκου για την αγαστή συνεργασία τους.

Θα ήθελα, επίσης, να ευχαριστήσω τη Γεωγραφική Υπηρεσία Στρατού, στην οποια υπηρετώ, καθώς μου έδωσε τη δυνατότητα να εργάζομαι και να αναπτύσσομαι σε μια Υπηρεσία υψηλού επιστημονικού επιπέδου και κύρους.

Θερμές ευχαριστίες εκφράζω και προς το Ίδρυμα Ωνάση για την οικονομική στήριξη που μου παρείχε με την επιλογή μου ως υπότροφου του Ιδρύματος. Ιδιαίτερη μνεία απευθύνω προς το προσωπικό του Ιδρύματος για την άψογη συνεργασία που είχαμε σε όλη τη διάρκεια του Προγράμματος Υποτροφίας.

Αναντίλεκτα, θα ήθελα να ευχαριστήσω τους γονείς μου, Σταύρο και Δήμητρα, και τα αδέλφια μου, Δημήτρη και Μάριο, για την παντοτινή τους αγάπη, καθώς υπήρξαν ανέκαθεν στήριγμα στις δυσκολίες μου.

Ακόμη, επιθυμώ να ευχαριστήσω θερμά τον πεθερό μου Κώστα και την πεθερά μου Αναστασία, καθώς βρίσκονται πάντα στο πλευρό της οικογένειάς μου, στηρίζοντάς την ανά πάσα στιγμή.

Τέλος, ιδιαίτερα, θα ήθελα να ευχαριστήσω από τα βάθη της καρδιάς μου τη σύζυγό μου Αντωνία, για την αμέριστη συμπαράσταση καθ' όλη τη διάρκεια των σπουδών μου (και όχι μόνο), τη συνεχή ενθάρρυνση, τη συναισθηματική στήριξη, την αδιαπραγμάτευτη εμπιστοσύνη, αλλά και τη στήριξη να φέρω αυτό το έργο εις πέρας.

Αφιερωμένο στη Γυναίκα της ζωής μου… ♥

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

A 3D city model is considered as the digital representation of a city/ urban area that may decompose into its objects/ elements such as buildings, roads, railways, terrain, water, vegetation etc., with clearly defined semantics, spatial and thematic properties. Depending on the level of detail (LoD), these objects may further decompose into more detailed features. The OGC standard CityGML, optimally allows integration of the diversified geoinformation of the aforementioned elements and provides multiple resolution at different LoDs. Since 2008, it has been an international OGC standard for representing and exchanging a 3D city model while in 2012, version 2.0 of this standard was published. CityGML represents the geometrical, semantic, and visual aspects of 3D city models and, for this reason, it is considered as an optimal standard for the representation of 3D city models. However, the structure of the CityGML standard is rather complex in order to support all these capabilities. Initially, CityGML was designed for the representation of 3D city models and not for presenting or visualizing 3D city models directly on the web. Therefore, the retrieval of the available semantic features from this standard, by implementing interoperable approaches without the need for specific knowledge, is a challenge, thus constituting the main research question of this work. Achieving this CityGML data retrieval is structured on the basis of interoperability, easy-to-use, semantics and non-expert user.

The current dissertation is structured in six chapters in order to address the research question raised above and the resulting sub-questions. First, the available research works and studies focusing on retrieving CityGML data are examined. Then, the solution of the REST approach is presented and compared with other state-of-the-art technologies, and finally, the CityGML RESTful Web service is conceptually designed and presented as a new approach for retrieving CityGML data based on their semantic characteristics.

Chapter 2 presents the relevant research work that focuses on the CityGML data retrieving utilizing tile or hierarchical-based or Web service-based approaches. Initially, the file-based formats such as X3D, JSON, KML and glTF have been further studied. Next, the OGC I3S and OGC 3D tiles are further examined as they provide a good solution in relation to the literature research. Next, taking into account the complex structure of the CityGML

standard and the need to retrieve data from distributed sources, the adoption of the available OGC Geospatial Web services are examined, such as OGC 3DPS and OGC WFS. Also, the extension of the OGC WFS, as well as the integration of the RESTful service architecture on top of OGC WFS are further examined.

The third Chapter of this dissertation studies the interoperable and easy-to-use information retrieval of CityGML based on its semantic characteristics using non-OGC Web services, such as SOAP and REST. Additionally, the REST is further compared with new state-of-the-art technologies that can be adopted as CityGML data retrieval mechanism, such as GraphQL and Falcor. Next, the solution of REST approach is presented and several principles and constraints in respect to the RESTful implementation are described. Thereafter, several principles and guidelines are provided with regard to the CityGML RESTful Web service and finally, the conceptual design of its core resources is presented such as "citymodels" and "gmlid".

Chapters 4 and 5 focus on the presentation and description of the conceptual design of CityGML RESTful Web service, which is a new approach and proposal of the current dissertation. So, taking into account the CityGML architecture, the CityGML structure is more semantic than geometric, and therefore the retrieval of the data has to be achieved mainly in compliance with the CityGML's semantic information. From the five components of the CityGML's architecture, only the component of the thematic modules defines the semantic features of CityGML. Therefore, these thematic modules are defined as the main resources of the CityGML RESTful Web service. However, apart from the above-mentioned resources, some extra main resources are also defined to make easier accessing their available semantic features. Since CityGML adopts the multi-scale modelling in five different LoDs, the same object may be simultaneously represented in different LoDs, enabling the analysis and visualization of the same object with regard to different resolution. However, LoD is considered vital not only in the geometric determination of the level of detail, but also in the semantics. By increasing the LoD, the semantic richness of CityGML increases respectively. Therefore, this semantic enrichment of each of the thematic modules is retrieved by implementing a variety of sub-resources. Thus, some of the main resources have LoD-based sub-resources and hence, their semantic retrieval is available based on the LoD, while, some resources are LoD-independent with no differentiation regarding their semantic sub-resources from one LoD to another.

More specifically, the fourth Chapter deals with the conceptual design of the LoD-based thematic resources of the CityGML RESTful Web service. In this direction, the "bldg", "tun" and "brid" main resources and their respective child resources are presented. These resources refer to the respective building, bridge and tunnel modules of the CityGML 2.0. Additionally, for each of these resources, various case studies using semantic requests are exploited and presented.

The conceptual design of the rest of the main resources of the CityGML RESTful Web service are presented in Chapter 5. These resources are mainly LoD-independent thematic resources and therefore, they are enriched with semantic characteristics either independently of LoD or from LoD2 and above without any different from one level to another.

Finally, Chapter 6 concludes this research work by discussing the findings of the previous chapters and responding to the sub-research questions formulated to address the aim of this dissertation. Suggestions for future research works are discussed, aiming at making this approach an OGC standard, and on upgrading it so that the upcoming version 3 of CityGML can be fully supported.

ΠΕΡΙΛΗΨΗ

Το τρισδιάστατο μοντέλο πόλης θεωρείται η ψηφιακή αναπαράσταση μιας πόλης που μπορεί να αποσυντεθεί σε ένα σύνολο αντικείμενων όπως κτήρια, δρόμοι, σιδηρόδρομοι, εδάφη, νερό, βλάστηση κλπ. με σαφώς καθορισμένη σημασιολογία, καθώς και χωρικές και θεματικές ιδιότητες. Αναλόγως το επίπεδο λεπτομέρειας, τα εν λόγω αντικείμενα μπορούν να αποσυντεθούν περαιτέρω σε πιο λεπτομερή χαρακτηριστικά. Το CityGML, που αποτελεί πρότυπο OGC, επιτρέπει την βέλτιστη ενσωμάτωση της ποικιλόμορφης γεωπληροφορίας των προαναφερθέντων στοιχείων παρέχοντας διαφορετική ανάλυση της πληροφορίας τους σε διαφορετικά επίπεδα λεπτομέρειας. Από το 2008 αποτελεί πρότυπο OGC για την αναπαράσταση και την ανταλλαγή 3D δεδομένων πόλης, ενώ από το 2012 βρίσκεται στην έκδοση 2.0. Επιπλέον, το CityGML αντιπροσωπεύει τις γεωμετρικές, σημασιολογικές και οπτικές πτυχές των 3D μοντέλων πόλης και, ως εκ τούτου, θεωρείται το καταλληλότερο πρότυπο για την αναπαράσταση τρισδιάστατων μοντέλων πόλης. Ωστόσο, προκειμένου να υποστηρίξει όλες τις προαναφερθείσες δυνατότητες, διαθέτει αρκετά πολύπλοκη δομή. Επιπλέον, ο βασικός στόχος σχεδίασης του CityGML είναι η αναπαράσταση του τρισδιάστατου μοντέλου πόλης και όχι η οπτικοποίησή του απευθείας στο διαδίκτυο. Επομένως, η δυνατότητα ανάκτησης όλων των διαθέσιμων σημασιολογικών πληροφορίων από το εν λόγω πρότυπο με την χρήση διαλειτουργικών προσεγγίσεων και χωρίς την ανάγκη ύπαρξης εξειδικευμένης γνώσης, αποτελεί σημαντική πρόκληση και δημιουργεί το βασικό ερευνητικό ερώτημα για τη διατριβή. Συγκεκριμένα, η ανάκτηση των δεδομένων του CityGML πρέπει να επιτευχθεί με γνώμονα τη διαλειτουργικότητα (interoperability), τη σημασιολογική προσέγγιση (semantically) και την εύκολη προσπέλαση /χρήση (easy-to-use), ακόμη και από μη ειδικούς (non-expert users).

Το περιεχόμενο της παρούσας διατριβής διαρθρώνεται σε έξι κεφάλαια, με στόχο την παροχή ολοκληρωμένων απαντήσεων στα ερευνητικά ερωτήματα που προέκυψαν από την προαναφερθείσα πρόκληση. Αρχικά, εξετάζονται οι διαθέσιμες έρευνες που εστιάζουν στην ανάκτηση δεδομένων CityGML. Στη συνέχεια, γίνεται παρουσίαση της προσέγγισης REST, η οποία στη συνέχεια συγκρίνεται με σύγχρονες τεχνολογίες. Τέλος, γίνεται αναλυτική παρουσίαση της CityGML RESTful Web service, που αποτελεί προτεινομένη λύση της παρούσας διατριβής ώστε να επιτευχθεί η ανάκτηση δεδομένων CityGML με βάση τα σημασιολογικά τους χαρακτηριστικά.

Το δεύτερο κεφάλαιο παρουσιάζει και αξιολογεί διάφορες έρευνες που επικεντρώνονται στην ανάκτηση δεδομένων CityGML χρησιμοποιώντας πληθώρα προσεγγίσεων όπως με πλακάκια (tile-based), ιεραρχικές (hierarchical-based) και διαδικτυακές υπηρεσίες. Αρχικά, μελετήθηκαν file-based μορφότυπα όπως X3D, JSON, KML και glTF. Στη συνέχεια τα πρότυπα OGC I3S και OGC 3D tiles μελετήθηκαν περαιτέρω καθώς παρέχουν αρκετά καλή λύση με βάση τις υπάρχουσες έρευνες. Επιπλέον, λαμβάνοντας υπόψη την περίπλοκη δομή του CityGML και την ανάγκη ανάκτησης δεδομένων από κατανεμημένες πηγές, εξετάστηκε η υιοθέτηση των διαθέσιμων OGC γεωχωρικών υπηρεσιών διαδικτύου, που στο πλαίσιο του τρισδιάστατου χώρου είναι τα OGC 3DPS και OGC WFS. Επίσης, αναφορικά με το OGC WFS, εξετάζεται περαιτέρω τόσο η επέκταση του όσο και η ενσωμάτωση RESTful διαδικτυακής υπηρεσίας ως βασικός οδηγός χρήσης του.

Το τρίτο κεφάλαιο αυτής της διατριβής μελετά τη διαλειτουργική και εύχρηστη ανάκτηση CityGML πληροφοριών με βάση τα σημασιολογικά χαρακτηριστικά τους χρησιμοποιώντας διαδικτυακές υπηρεσίες που δεν αποτελούν πρότυπα OGC, όπως SOAP και REST. Επιπροσθέτως, η REST αρχιτεκτονική συγκρίνεται περαιτέρω με νέες τεχνολογίες αιχμής που μπορούν να υιοθετηθούν ως μηχανισμός ανάκτησης δεδομένων CityGML, όπως GraphQL και Falcor. Έπειτα, παρουσιάζεται η προσέγγιση REST ως προτεινόμενη λύση και, επιπλέον, περιγράφονται διάφορες αρχές και περιορισμοί που αναφέρονται στη RESTful υλοποίηση. Στη συνέχεια, παρέχονται αρχές και οδηγίες αναφορικά με την CityGML RESTful διαδικτυακή υπηρεσία και τέλος, αναλύεται ο εννοιολογικός σχεδιασμός των πόρων του πυρήνα της, όπως "citymodels" και "gmlid".

Τα Κεφάλαια 4 και 5 εστιάζουν στην αναλυτική περιγραφή και παρουσίαση του εννοιολογικού σχεδιασμού της CityGML RESTful διαδικτυακής υπηρεσίας, η οποία αποτελεί μια νέα προσέγγιση και πρόταση της τρέχουσας διατριβής. Συνεπώς, λαμβάνοντας υπόψη την αρχιτεκτονική του CityGML, η δομή του είναι περισσότερο σημασιολογική παρά γεωμετρική και επομένως η ανάκτηση των δεδομένων πρέπει να υλοποιηθεί κυρίως σύμφωνα με τις σημασιολογικές πληροφορίες του. Από τα πέντε στοιχεία της αρχιτεκτονικής του CityGML, μόνο το στοιχείο των θεματικών μοντέλων καθορίζει τα σημασιολογικά χαρακτηριστικά του CityGML. Συνεπώς, τα εν λόγω θεματικά μοντέλα καθορίζονται ως οι βασικοί πόροι της CityGML RESTful διαδικτυακής υπηρεσίας. Ωστόσο, εκτός από τους προαναφερθέντες πόρους, καθορίζονται κάποιοι επιπλέον βασικοί πόροι προκειμένου να διευκολυνθεί η πρόσβαση στα διαθέσιμα σημασιολογικά τους χαρακτηριστικά. Επιπροσθέτως, υιοθετεί τη μοντελοποίηση πολλαπλών κλιμάκων και υποστηρίζει πέντε διαφορετικά επίπεδα λεπτομέρειας. Στο CityGML, το ίδιο αντικείμενο δύναται να αναπαρασταθεί ταυτόχρονα σε διαφορετικά επίπεδα λεπτομέρειας, επιτρέποντας την οπτικοποίηση του ίδιου αντικειμένου σε διαφορετικά επίπεδα χωρικής ανάλυσης. Ωστόσο, το επίπεδο λεπτομέρειας θεωρείται ζωτικής σημασίας, τόσο στο γεωμετρικό προσδιορισμό των διαθέσιμων χαρακτηριστικών όσο και στο σημασιολογικό, και επομένως, η αύξηση του επιπέδου λεπτομέρειας εμπλουτίζει αντίστοιχα τα σημασιολογικά χαρακτηριστικά του CityGML. Ως εκ τούτου, η ανάκτηση του εκάστοτε σημασιολογικού εμπλουτισμού για κάθε διαθέσιμο θεματικό μοντέλο επιτυγχάνεται με την υιοθέτηση διαφόρων υπο-πόρων της CityGML RESTful διαδικτυακής υπηρεσίας. Ως αποτέλεσμα, ορισμένοι από τους βασικούς πόρους διαθέτουν υπο-πόρους που η διαθεσιμότητά τους βασίζεται στο επίπεδο λεπτομέρειας, ενώ ορισμένοι πόροι είναι ανεξάρτητοι από το επίπεδο λεπτομέρειας και επομένως δεν υπάρχει διαφοροποίηση στη διαθεσιμότητα των αντίστοιχων υπο-πόρων τους από το ένα επίπεδο λεπτομέρειας στο άλλο.

Το τέταρτο κεφάλαιο ασχολείται με τον εννοιολογικό σχεδιασμό των LoD-based βασικών θεματικών πόρων της CityGML RESTful διαδικτυακής υπηρεσίας. Συγκεκριμένα, παρουσιάζονται οι βασικοί πόροι "bldg", "tun" και "brid" και οι αντίστοιχοι υπο-πόροι τους. Οι εν λόγω βασικοί πόροι αναφέρονται στα αντίστοιχα μοντέλα κτηρίων, γεφυρών και τούνελ του CityGML 2.0. Επιπροσθέτως, για κάθε έναν από αυτούς τους πόρους, παρουσιάζονται διάφορα παραδείγματα εφαρμογής σημασιολογικών αιτημάτων.

Ο εννοιολογικός σχεδιασμός των υπολοίπων βασικών πόρων της CityGML RESTful διαδικτυακής υπηρεσίας αναλύεται στο Κεφάλαιο 5. Οι συγκεκριμένοι πόροι είναι ανεξάρτητοι από το επίπεδο λεπτομέρειάς τους και επομένως, εμπλουτίζονται με τα ίδια σημασιολογικά χαρακτηριστικά είτε ανεξάρτητα του εκάστοτε επιπέδου λεπτομέρειας είτε από το επίπεδο λεπτομέρειας δύο και πάνω.

Τέλος, στο Κεφαλαίο 6 ολοκληρώνεται η μελέτη της τρέχουσας διατριβής συζητώντας τα ευρήματα των προηγούμενων κεφαλαίων και απαντώντας στα αντίστοιχα ερευνητικά ερωτήματα. Επίσης, υποβάλλονται προτάσεις για μελλοντική έρευνα, εστιάζοντας στην καθιέρωση της προτεινόμενης προσέγγισης ως πρότυπο OGC, καθώς επίσης και στην αναβάθμισή της προκειμένου να μπορεί να υποστηρίξει πλήρως την επερχόμενη έκδοση 3 του CityGML.

LIST OF ABBREVIATIONS

2D	Two Dimensions
3D	Three Dimensions
3DCIM	3D City Information Model
3DCityDB	3D City Database
3DPIE	3D Portrayal Interoperability Experiment
3DPS	3D Portrayal Service
ADE	Application Domain Extension
AGI	Analytical Graphics Inc
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
B3DM	Batched 3D Model
BIM	Building Information Model
CAD	Computer-Aided Design
CityGML	City Geography Markup Language
COBRA	Common Object Request Broker Architecture
COLLADA	COLLAborative Design Activity
CORS	Cross-Origin Resource Sharing
CQL	Common Query Language
CRS	Coordinate Reference System
CSV	Comma-Separated Values
CSW	Catalog Service for the Web
DBMS	Database Management System
DOM	Document Object Model
EPSG	European Petroleum Survey Group
FME	Feature Manipulation Engine
GDF	Geographic Data Files
GIS	Geographic Information Systems
glTF	GL Transmission Format
GML	Geography Markup Language
GMLID	GML identifier
GPU	Graphics Processing Unit
HATEOAS	Hypermedia as the Engine of Application State
HTML	HyperText Markup Language
HTTP (S)	HyperText Transfer Protocol (Secure)
I3S	Indexed 3D Scene Layer
IFC	Industry Foundation Classes

ISO	International Organization for Standardization
IT	Information Technology
JAXB	Java Architecture for XML Binding
JSON	JavaScript Object Notation
JSONP	JSON with Padding
KML	Keyhole Markup Language
KMZ	Zipped KML Format
LADM	Land Administrator Domain Model
LoD	Level of Detail
Oauth	Token Based Authentication
OGC	Open Geospatial Consortium
REST	Representational State Transfer
ROA	Resource-Oriented Architecture
RPC	Remote Procedure Call
SIG 3D	Special Interest Group 3D
SLPK	Scene Layer Package
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
TIN	Triangulated Irregular Network
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
VRML	Virtual Reality Markup Language
W3DS	Web 3D Service
WCS	Web Coverage Service
WebGL	Web Graphics Library
WFS	Web Feature Service
WMS	Web Map Service
WOA	Web-Oriented Architecture
WSDL	Web Services Description Language
WVS	Web View Service
X3D	Extensible 3D
X3DOM	pronounced X-Freedom
XML	eXtensible Markup Language

1. INTRODUCTION

1.1 Context

1.2 Problem Statement

1.3 Research Questions

1.4 Outline

CHAPTER 1: INTRODUCTION

1.1. Context

Many urban or environmental models are defined for supporting practitioners and stakeholders in their decision-making processes. Models which represent in three dimensions (3D) the geometric and semantic elements of city are called 3D city models. These 3D city models are increasingly used in different cities and countries for an intended wide range of applications beyond mere visualization (Pispidikis & Dimopoulou, 2019), providing further value and additional utility over two dimensions (2D) geo-datasets. Additionally, they are becoming ubiquitous for making decisions and for improving the efficiency of governance. Hence, the generation of complex 3D city models facilitates the better sophisticated understanding of the objects and their spatial interaction with their surrounding environment (Floros et al., 2017). The type and amount of data that can be integrated into a 3D city model rises dramatically, a condition that promotes the necessity all these data to be properly stored, edited, visualized and retrieved.

3D city models come in various versions. The most commonly used technologies for 3D city models are Geographic Information Systems (GIS), while on a building-scale Building Information Modelling (BIM) is mostly used. BIM and GIS refer to different spatial scales and modelling levels and thus, various data exchange standards, protocols and formats have been developed to serve the needs for each domain (Pispidikis et al., 2018). Currently, Industry Foundation Classes (IFC) and City Geography Markup Language (CityGML) are representative model standards for BIM and GIS, respectively. Even though other formats exist, they are the most widely studied and used exchange formats. Furthermore, they are also complete ontologies for building and city models that could contribute to the construction of the semantic web. As a result, focusing on 3D city models, CityGML is considered the optimal standard for the semantic, geometric and topological representation of a city. 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 (Gröger et al., 2012). However, although CityGML is considered as the most appropriate model for the representation of 3D city models, it is quite difficult to retrieve this data based on their semantic geometric and descriptive features.

The aforementioned issue was identified during the author's master thesis with the following topic (Pispidikis & Dimopouloy, 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"

The objective of the abovementioned thesis was the development of a 3D WebGIS application in order to successfully retrieve and visualize CityGML data in accordance with their respective geometric and semantic characteristics in all Levels of Detail (LoD). Although there have been several research projects on the visualization of 3D city models utilizing the CityGML standard, there was no solution regarding the semantic retrieval of this data. To this purpose, a suitable PHP class called cityDBWrapper was developed and hence, the data retrieval from PostGIS was achieved, based on both semantic characteristics and LoDs of CityGML.

However, although the implementation of this approach provides a good solution for semantically retrieving CityGML data, some issues are presented to be resolved relating to the complex structure of the CityGML and the need to retrieve data from distributed sources without requiring specific knowledge of the source (CityGML) and of the proposed Appliation Programming Interface (API).

1.2. Problem Statement

A 3D city model is considered as the digital representation of a city that may decompose into its objects (such as buildings, roads, railways, terrain, water, vegetation etc.) with clearly defined semantics, spatial and thematic properties. Depending on the levels of detail, these objects may further decompose into more detailed features. The OGC standard CityGML, optimally allows integration of the diversified geoinformation of the aforementioned elements and provides multiple resolutions at different LoDs. However, the structure of the CityGML is rather complex for supporting all these capabilities. Therefore, the retrieval of the available semantic features from this standard by implementing interoperable approaches without the need for advanced knowledge, is a challenge.

1.3. Research Questions

Taking into consideration the abovementioned problem statement and the respective research studies, the following research questions should be investigated and answered in the context of the current dissertation:
Thus, the core research question is:

How the **interoperable** and **easy-to-use** information retrieval of a city could be **semantically** achieved by **non-expert** user?

Thereafter, the following sub-research questions arise:

(1) Can the existing 3D graphics or data exchange formats be utilized as a means of semantically retrieving CityGML data?
(2) Can the existing OGC Geospatial Web services be utilized as a means of semantically retrieving CityGML data?
(3) What is the most appropriate architecture type of a web service for achieving the easy-to-use information retrieval of a city?
(4) How could CityGML data be semantically retrieved by users without knowledge of the source?

1.4. Outline

Chapter 2 examines various approaches for retrieving and visualizing CityGML data. Initially, the tile and hierarchical-based approaches using file-based formats such as X3D, JSON, KML and glTF have been further studied and research. Next, the OGC I3S and OGC

3D Tiles are also examined. Finally, the available OGC Geospatial Web services are studied which, in the context of 3D, there are the 3DPS and the WFS.

Chapter 3 examines the use of non-OGC Web services for the interoperable and easy-to-use information retrieval of a CityGML based on its semantic characteristics. For this purpose, The SOAP and REST Web services are further studied and compared. Also, the REST are compared with new state-of-the-art technologies that can be adopted as a CityGML data retrieval mechanism such as GraphQL and Falcor. Thereafter, several principles and guideline are addressed with regard to the CityGML RESTful Web service and finally, the conceptual design of the "citymodels" and "gmlid" resources is presented.

Chapter 4 describes the conceptual design of the LoD-based thematic resources of the CityGML RESTful Web service. More specifically, the "bldg", "tun" and "brid" resources and their respective child resources are presented. Also, for each of these resources, various case studies using semantic requests are presented.

Chapter 5 presents the conceptual design of the rest of the main resources of the CityGML RESTful Web service which are mainly LoD-independent thematic resources. Namely, these resources are enriched with semantic characteristics either independently of LoD or from LoD2 and above without any difference from one level to another. Hence, the thematic resources with similar availability in all LoD as well as the thematic resources with similar availability from LoD2 and above are presented.

Finally, Chapter 6 concludes the findings with respect to the core research question of the current dissertation and sets suggestions for future research.

2. RETRIEVING CITYGML INFORMATION

2.1 Tiled and Hierarchical-based CityGML retrieval

2.2 3D Web Portrayal Services

2.3 OGC Web Services for Sharing and Managing Raw Data

This chapter examines various approaches for retrieving and visualizing CityGML data. Initially, the tiled and hierarchical-based approaches using file-based formats such as X3D, JSON, KML and glTF have been further studied and investigated. Second, the OGC I3S and OGC 3D Tiles were also examined. Finally, the available OGC Geospatial Web services were studied which are, in the context of 3D, the 3DPS and the WFS.

In this chapter, the 1st and 2nd sub-research questions of the current dissertation are addressed:

(1) *Can the existing 3D graphics or data exchange formats be utilized as a means of semantically retrieving CityGML data?*

(2) *Can the existing OGC Geospatial Web services be utilized as a means of semantically retrieving CityGML data?*

This chapter utilizes the following papers:

(1) **Pispidikis** and Dimopoulou (2016)

(2) **Pispidikis** and Dimopoulou (2018)

(3) **Pispidikis**, Tsiliakou, Kitsakis, Athanasiou, Kalogianni, Labropoulos, and Dimopoulou (2018)

(4) Athanasiou, **Pispidikis** and Dimopoulou (2018)

(5) **Pispidikis** and Dimopoulou (2019)

(6) Chatzinikolaou , **Pispidikis** and Dimopoulou (2020)

2.1. Tiled and Hierarchical-based CityGML Retrieval

CityGML presents an efficient solution for the representation of 3D city models because it combines geometry and semantics in a single data model. However, efficiently visualizing or retrieving 3D geometry and semantic information stored in CityGML is complex. It should be noted that a number of desktop viewers are available for the local visualization of CityGML data such as FZK Viewer and Feature Manipulation Engine (FME) Data Inspector. However, the visualization of CityGML models on the web is still a challenging area since CityGML is designed for the representation of 3D city models and not for presenting or visualizing 3D city models directly on the web (Ohori, Biljecki, Kumar, Ledoux, & Stoter, 2018). Hence, several research works have been done on the aforementioned challenge focusing on retrieving CityGML data implementing either tiled and Hierarchical-based or Web-service-based approaches.

2.1.1. 3D graphics and data exchange formats

2.1.1.1. Extensible 3D (X3D)

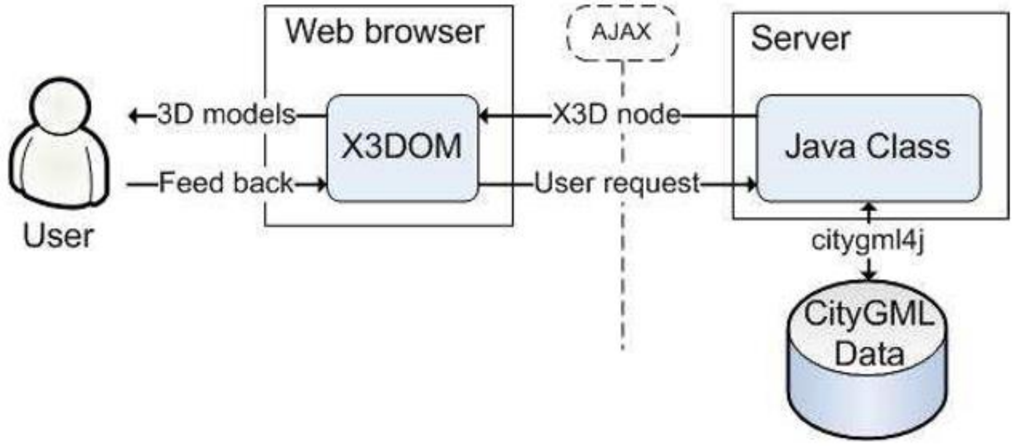
In the context of the implementation of 3D graphics, Mao and Ban (2011) developed a framework for the visualization of 3D city models (Figure 2-1). As data source the CityGML was used, which turned into an Extensible 3D[1] (X3D) scene and finally it was visualized on the web utilizing the pronounced X-Freedom (X3DOM)[2]. Specifically, CityGML data were analyzed and converted to Java Classes, representing various city objects such as buildings, streets, etc. The said conversion was implemented by the use of Citygml4j[3] Application

[1] **X3D** is an XML-based, open 3D data format that is used to represent 3D scenes in a web environment and is the successor to Virtual Reality Modelling Language (VRML).

[2] **X3DOM** is a framework for integrating X3D scenes as HTML5 Document Object Model (DOM) elements, which are rendered via WebGL without additional plugins.

[3] **Citygml4j** is an open source Java class library and API for facilitating work with the CityGML. Citygml4j makes it easy to read, process and write CityGML datasets, and to develop CityGML-aware software applications.

Programming Interface (API). Then, the respective scenes were generated in accordance with geometric or semantic information.



-

Figure 2-1: File-based approach for the visualization of CityGML over the web

(Mao & Ban, 2011)

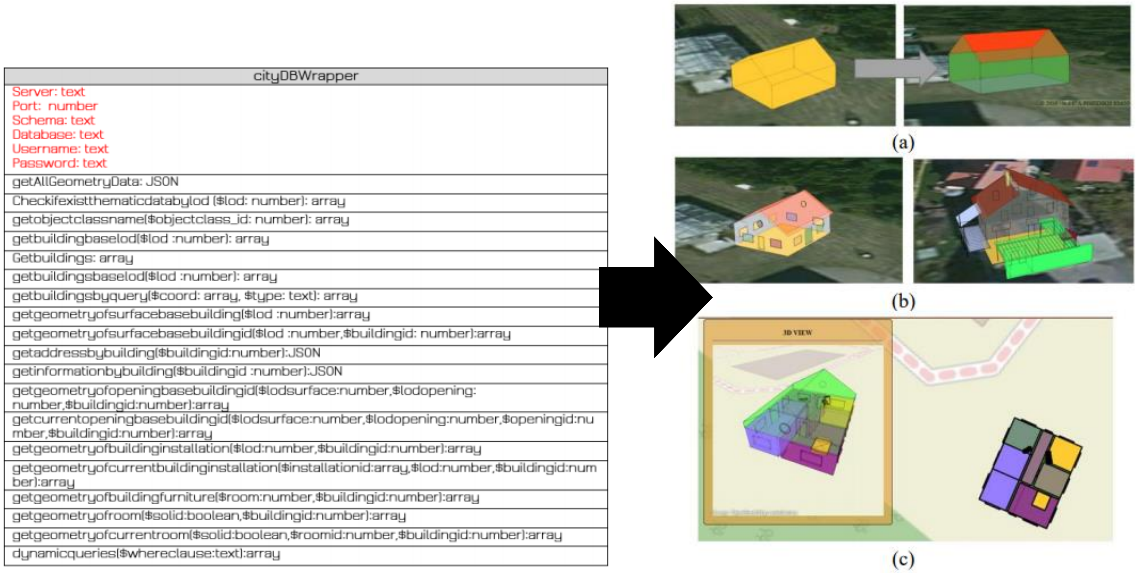
2.1.1.2. JavaScript Object Notation (JSON)

LSIS (Laboratoire des sciences de l' information et des systemes) laboratory focused on the representation of CityGML buildings carried out three tests. In the first test, the entire CityGML file was fetched from a WFS server on HyperText Markup Language (HTML) thick client based on C++. In the second test, the CityGML file was first processed on a server using Java Architecture for XML Binding (JAXB) parser. Consequently, only the required part can be fetched on the client. In the last test, the CityGML stream was replaced with a JavaScript Object Notation (JSON) stream. This choice was made taking into account that the latter can be more easily portrayed on the web using Three.js API, which utilizes the WebGL[4] technology (Schilling, Hagedorn, & Coors, 2012). Extending to the last test is the approach of Gesquiere and Manin (2012), who adopted the tile-based approach to work on CityGML files. The CityGML file was broken into several tiles and each

[4] **WebGL** (Web Graphics Library) is a JavaScript API for rendering high-performance interactive 3D and 2D graphics within any compatible web browser without the use of plug-ins.

tile was transformed into JSON, which was stored on the server. Hence, the client made requests to the server on the basis of specific tile and consequently, the server responded with JSON file for that tile. Prandi et al. (2013), following the tile-based approach, developed a framework in the context of a project called iSCOPE (interoperable Smart City Services through an Open Platform for urban Ecosystems). Specifically, they separated the CityGML files into tiles, storing them to the Server and finally, the Client can make requests based on the said requests. As a result, the progressive visualization was achieved.

However, the implementation of the above-mentioned studies does not provide solution regarding the semantic retrieval of CityGML data without need for knowledge of the source. Consequently, Pispidikis and Dimopoulou (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 (Figure 2-2).



```
cityDBWrapper
Server: text
Port:  number
Schema: text
Database: text
Username: text
Password: text
getAllGeometryData: JSON
Checkifexistthematicdatabylod ($lod: number): array
getobjectclassname($objectclass_id: number): array
getbuildingbaselod($lod :number): array
Getbuildings: array
getbuildingsbaselod($lod :number): array
getbuildingsbyquery($coord: array, $type: text): array
getgeometryofsurfacebasebuilding($lod :number):array
getgeometryofsurfacebasebuildingid($lod :number,$buildingid: number):array
getaddressbybuilding($buildingid:number):JSON
getinformationbybuilding($buildingid :number):JSON
getgeometryofopeningbasebuildingid($lodsurface:number,$lodopening:
number,$buildingid:number):array
getcurrentopeningbasebuildingid($lodsurface:number,$lodopening:number,$openingid:nu
mber,$buildingid:number):array
getgeometryofbuildinginstallation($lod:number,$buildingid:number):array
getgeometryofcurrentbuildinginstallation($installationid:array,$lod:number,$buildingid:num
ber):array
getgeometryofbuildingfurniture($room:number,$buildingid:number):array
getgeometryofroom($solid:boolean,$buildingid:number):array
getgeometryofcurrentroom($solid:boolean,$roomid:number,$buildingid:number):array
dynamicqueries($whereclause:text):array
```

Figure 2-2: PHP class for semantic retrieve LoD2 (a), LoD3 (b) and LoD4 (c) CityGML data (Pispidikis & Dimopoulou, 2016)

2.1.1.3. Keyhole Markup Language (KML)

The increased focus on HTML5 and WebGL solutions leads to the development of an entire framework for 3D geospatial data visualizations such as Cesium and iTowns. These frameworks feature an open source JavaScript and WebGL-based virtual globe and map engine that can display terrain, image, and 3D models. The Keyhole Markup Language (KML) and the GL Transmission Format (glTF) are natively supported by these frameworks.

KML is an XML grammar language used to encode and transport representations of geographical data for display in a Web browser. KML was originally created as a file format for Keyhole's Earth Viewer, which later emerged as the Google Earth application allowing users to overlay their own content on top of the basemaps. In 2007, Google submitted KML to the OGC and in 2008 was adopted as OGC standard (Wilson, 2008). KML files are often distributed in Zipped KML Format (KMZ) files, which are zipped files that include KML along with its associated images and icons. According to the KML specification, the tile-based retrieval of the data can be achieved implementing the NetworkLink element (Figure 2-3).

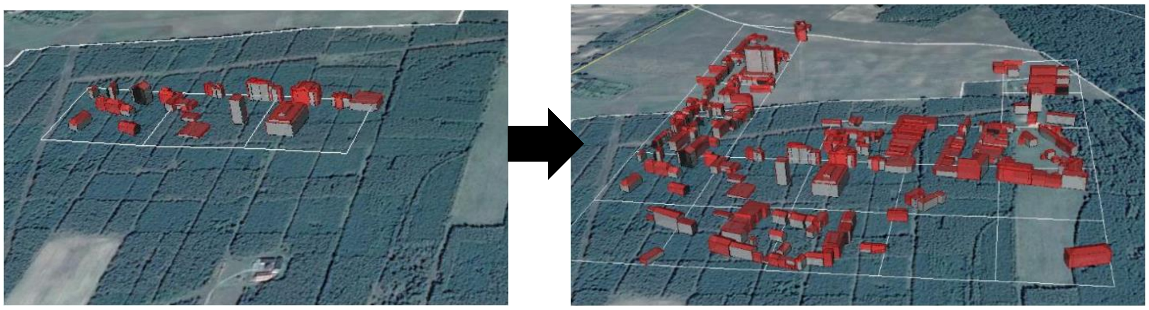


Figure 2-3: Tile-based retrieval using NetworkLinks

The aforementioned tile-based mechanism was implemented by Chaturvedi (2014). More specific, the 3DCityDB Importer/Exporter[5] tool was used and the CityGML files were

[5] **3DCityDB Importer/Exporter** is a java-based front-end for the 3D City Database and allows for high-performance loading and extracting 3D city model data. Specifically, the supported import and export operations are the following: import of CityGML models;export data as CityGML models; export data in KML/COLLADA/glTF format; export data as spreadsheets.

imported into a 3D city database[6]. Next, these datasets were exported in KMZ files based on specified number of tiles and their respective length. Further, the reference of the KMZ files, in accordance with the tiles, is given in master KML file using NetworkLinks.

Additionally, Prandi et al. (2015) involved with the 3D web visualization of huge CityGML models, which were originally stored in the Database in compliance with 3D city database schema. Thereafter, in order the visualization of their data to be achieved and, in addition, their thematic features to be able to be searched, the following procedures were implemented: firstly, for visualization purpose the data was exported to Keyhole Markup Language/ COLLAborative Design Activity (KML/COLLADA) format together with the specific CityGML ID of the feature; secondly, the data was retrieved from the 3D city database utilizing the OGC WFS server using the CItyGML ID as query attributes.

Chaturvedi et al. (2015) presented a Web based 3D client, which has been developed on top of WebGL based Cesium virtual globe utilizing the following technologies: ExtJS JavaScript-based web framework and HTML5. The highlighted features of the said client are the data exploration, the managing interaction and the queries based on the attributes of the data. The visualization of the data was achieved using KML/COLLADA files and JSON encoded data.

2.1.1.4. GL Transmission Format (glTF)

The glTF is an API-neutral runtime asset delivery format that bridges the gap between 3D content creation tools and modern graphics applications by providing an efficient, extensible, interoperable format for the transmission and loading of 3D content. This format combines an easily parsable JSON scene description with one or more binary files representing geometry, animations, and other rich data. Binary data is stored in such a way that it can be loaded directly into Graphics Processing Unit (GPU) buffers without additional

[6] **3D City Database** is a free geo database to store, represent and manage virtual 3D city models on top of a standard spatial relational database. This database schema implements the CityGML standard with semantically rich and multi-scale urban objects facilitating complex analysis task, far beyond visualization.

parsing or other manipulation (Figure 2-4). Implementing this approach, glTF is able to faithfully preserve full hierarchical scenes with nodes, meshes, cameras, materials, and animations, while enabling efficient delivery and fast loading (Khronos Group, 2019). The implementation of glTF format for streaming CityGML 3D city models was described by Schilling et al. (2016). They concluded that using formats such as X3D, KML/COLLADA or glTF makes the rendering process using existing visualization frameworks particularly simple. However, these pure graphics formats cannot directly store CityGML's semantic information. Similarly, Ohori et al. (2018) noted that the visualization of CityGML over the web using commonly 3D graphics requires the separation of geometric information from semantic information. Consequently, the rich semantics of CityGML are often lost.

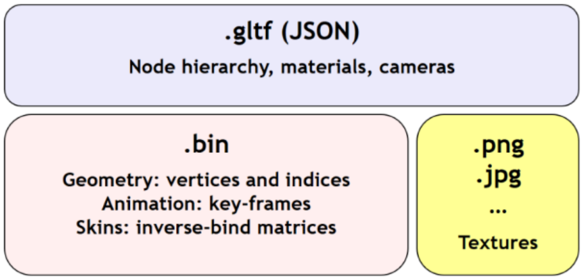

.gltf (JSON)
Node hierarchy, materials, cameras
.bin
Geometry: vertices and indices
Animation: key-frames
Skins: inverse-bind matrices
.png
.jpg
...
Textures

Figure 2-4: Valid glTF asset

(Khronos Group, 2019)

2.1.2. OGC 3D Tiles

In the context of GIS, properties of objects, e.g. buildings, are inherently part of the virtual representation and must be accessible either as embedded attributes, as separate table or via supplementary database queries. The common 3D formats such as X3D, COLLADA and glTF have no designated place for storing additional object information. The format that merges glTF assets and attributes was developed and shared under the umbrella of the OGC 3D tiles[7] (Cozzi, 2019) and called B3DM (Batched 3D Model). This format introduces the concept of batches for identifying objects and assigning properties

[7] **OGC 3D Tiles** is designed for streaming and rendering massive 3D geospatial content such as Photogrammetry, 3D Buildings, BIM/CAD, Instanced Features, and Point Clouds. It defines a hierarchical data structure and a set of tile formats which deliver renderable content.

such as unique ID, feature type and custom attributes. Each object property can be used for highlighting, for showing/hiding specific object, for custom styling based on attributes and for querying web services for retrieving additional information based on the ID (Figure 2-5). However, for embedding and preserving all the available semantic features of CityGML in Cesium, all data must be made available as 3D Tiles layer and converted into B3DM. Due to different concepts regarding spatial data representation and basic structuring, a series of processing steps must be performed that go away beyond a simple format conversion.

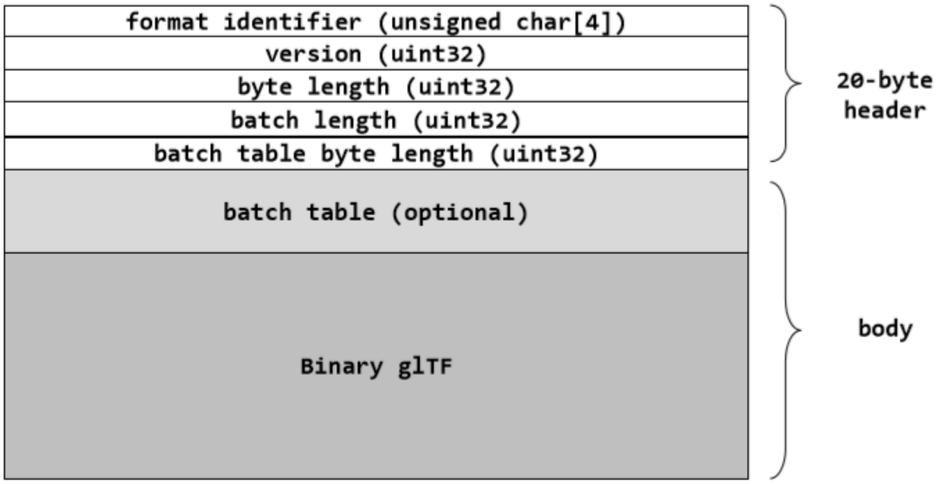


Figure 2-5: layout of a B3DM

(Schilling et al., 2016)

2.1.3. OGC Indexed 3D Scene Layer (I3S) and Scene Layer Package

I3S (Indexed 3D Scene Layer) was released to the community by ESRI as a format for packaging and streaming large, heterogeneously distributed 3D data sets and was adopted in 2017 as an OGC standard (Reed & Belayneh, 2017). The I3S is declarative and extendable, and can be used to represent different types of 3D data such as 3D objects, integrated mesh, point, point cloud and building scene layer. It is encoded using JSON and binary ArrayBuffers (see ECMAScript 2015 known as ES6). The main goal of this standard is to enable streaming large 3D datasets with high performance and scalability and hence, it is designed from the ground up to be cloud, web and mobile friendly. Also, it is based on JSON, REST and modern web standards, making it easy to handle, parse, and render by web and mobile clients. Currently, the scene layers can be consumed from any ArcGIS applications such as ArcGIS Pro, ArcGIS Earth, ArcGIS online, CityEngine etc. either as service or local scene layer package files (SLPK).

The I3S standard was implemented by Pispidikis et al. (2018). Specifically, they combined different 3D modelling methodological tools and techniques (Figure 2-6) to develop a semantically enriched 3D campus model that can be used for navigation and maintenance.

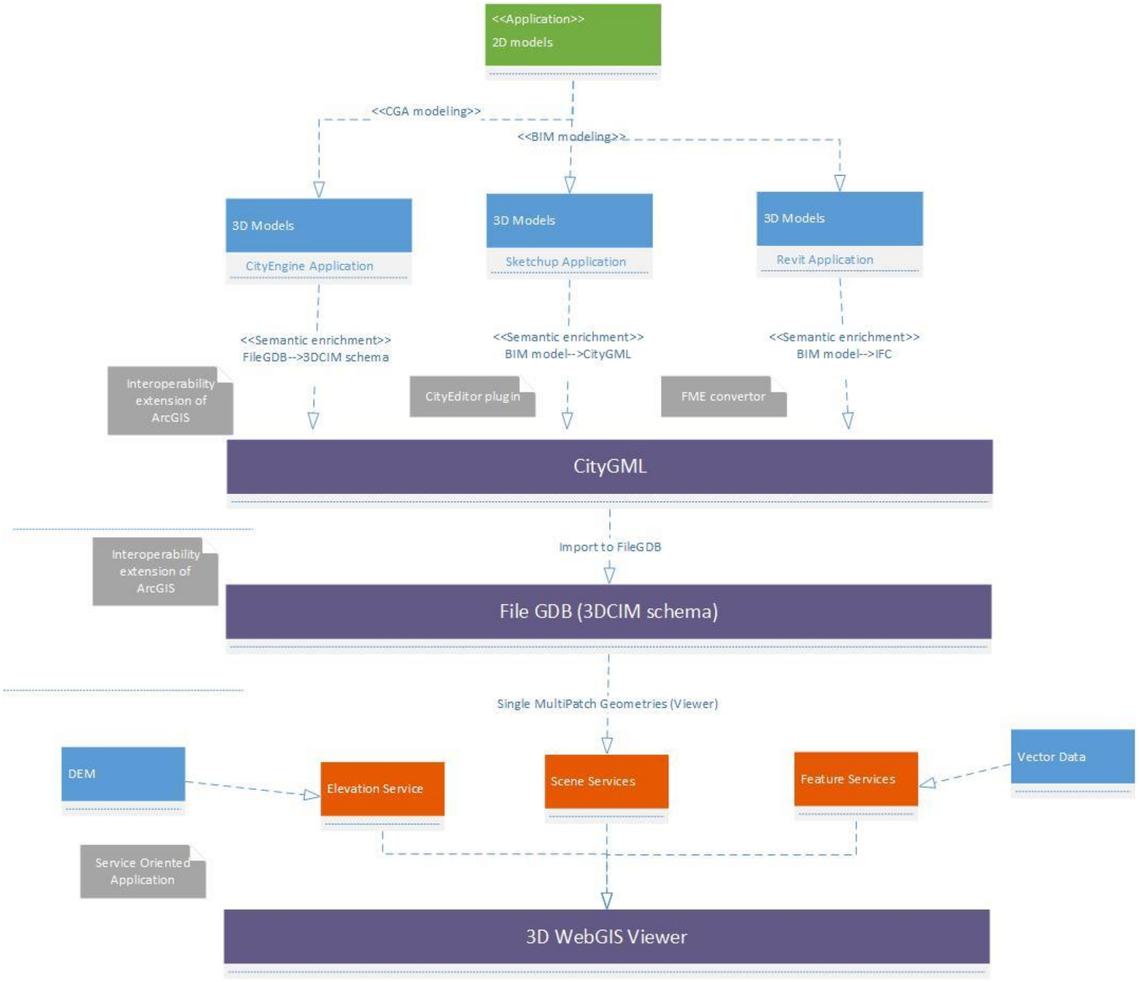


Figure 2-6: Methodological tools and techniques

(Pispidikis et al., 2018)

For buildings' modeling, two different modeling approaches were implemented (procedural & BIM-based modelling), using several software such as CityEngine, Trimble SketchUp Pro and Autodesk Revit. Thereafter, each developed model was semantically enriched, to be represented in LoD1, LoD2 and LoD3 of CityGML standard and then, imported to a file Geodatabase, based on the 3D City Information Model (3DCIM)[8] schema. The 3DCIM and the CityGML are considered complementary and hence, several tools have been developed to achieve interoperability for these models (Reitz et al., 2014). Next, the file Geodatabases were converted to SLPKs and published as ArcGIS Scene Services (Figure 2-7). However, for preserving all the available semantic features of CityGML in all LoDs, all semantic features must be embedded as data attributes in each file Geodatabase according to the 3DCIM schema. The said procedure is significantly complex to be implemented for large scale city models.

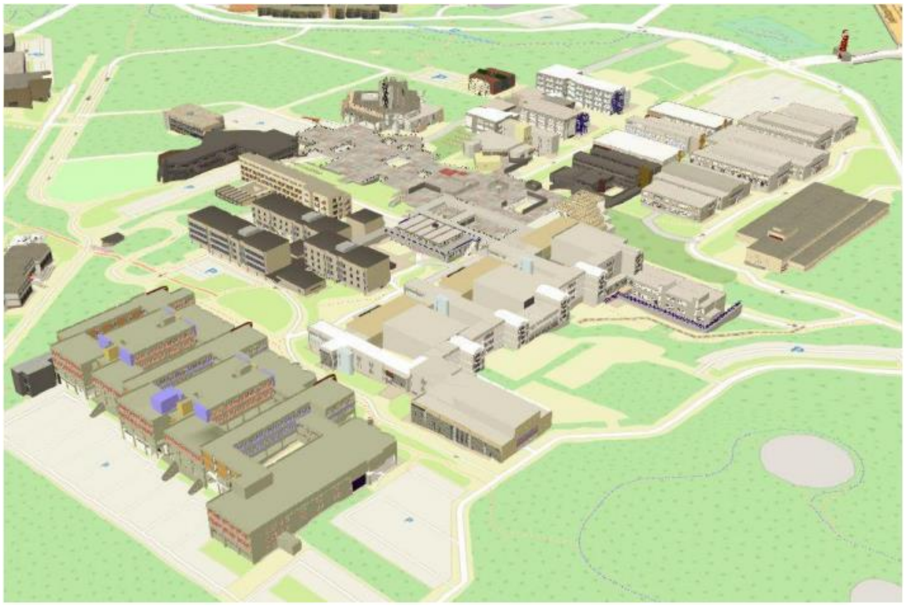


Figure 2-7: 3D NTUA Campus overview using ArcGIS Scene Services

(Pispidikis et al., 2018)

[8] **3DCIM** is the commercial solution of the semantically enriched database schema, developed by ESRI, aiming to provide compact and yet simple in structure, information model.

2.2. 3D Web Portrayal Services

Taking into consideration the aforementioned studies and research (see section 2.1) regarding the tiled and hierarchical-based approaches for retrieving and visualizing CityGML data using file-based formats, there have been several issues. The visualization of CityGML over the web using commonly 3D graphics requires the separation of geometric information from semantic information and hence, the rich semantics of CityGML are often lost. Additionally, although the OGC I3S and OGC 3D Tiles provide partial solution, the procedure to generate these files from CityGML source, retrieving all semantic features, is complex. Last but foremost, the implementation of these solutions is not suitable in terms of interoperability. Therefore, taking into account the complex structure of CityGML and the need to retrieve data from distributed sources addressing interoperability issues, adoption of Web service technology is required.

2.2.1. Technology of Web Services

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, 1999-2020).

2.2.1.1. The benefits of Web Services

According to the OGC glossary of terms, interoperability is "*the Capability to communicate, execute programs, or transfer data among various functional units in a manner that requires the user to have little or no knowledge of the unique characteristics of those units*".

Additionally, interoperability, in the context of the OpenGIS specification, is a software component operating reciprocally (working with each other) to overcome tedious batch conversion tasks, import/export obstacles, and distributed resources access barriers imposed by heterogeneous processing environments and heterogeneous data.

The main goal of web services is to exchange information 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). Additionally, the Web services facilitate the ability to build composite applications based on the heterogeneous services operating across many different platforms. Namely, whatever programming language is used to implement a Web service, whatever operating system it runs on, and whatever Web application server it is deployed on, none of these affects how clients can consume the service. Thus, Web services and their clients are not tightly bound to one another. A Web service can be consumed by multiple clients, and a client can consume multiple Web services. Also, a Web service and its clients do not need to run on the same server, and they do not need to be compiled together. Additionally, developers have the freedom to choose whatever tools or programming language they desire. Furthermore, when a Web service is updated or a new version is released, the change only needs to be made on the server side. Thereafter, all clients consume the latest version. Also, there is no need to run installation or an update on each client computer providing a significant advantage of Web service over desktop programming components.

2.2.1.2. Geospatial Web Service Standards

An explosion of Web-based mapping applications followed the birth of WebGIS in 1993, and a small amount of WebGIS software products appeared on the market. However, these early technologies had limitations in both their internal architecture and in their integration with other information systems. Because of these limitations, Web GIS was underused, and its potential was not fully realized (Huang, 2002).

Over the years, the concepts, standards, and technology for implementing GIS interoperability have evolved through six stages: (1) data converters, (2) standard interchange formats, (3) open file formats, (4) direct-read APIs, (5) common features in a database management system (DBMS), and (6) integration of standardized Geospatial Web services (Fu & Sun, 2010). Geospatial Web services have become the heart of GIS, representing significant progress in distributed GIS. Additionally, they hide the complexity of GIS data and functionality, leaving it to be handled remotely on other servers, while exposing a Web programming interface for easy integration. Thus, the Information Technology (IT) can simply access mapping, data, and geoprocessing web services from a variety of sources without having to deal locally with the geospatial complexity. This capability gives GIS industry the ability to move beyond data conversion and convert installation into Web service-based interoperability. Realizing this opportunity, standards bodies such as OGC and International Organization for Standardization (ISO) have defined a series of Web services standards. With these standards, GIS application are not tied to a specific software vendor. Organizations can manage data using the methods and formats best suited to their needs while exposing Web service interface that conform to specific open standards. Thereafter, other users can use these services regardless of which vendors are behind the services. Therefore, OGC developed and implemented several Geospatial Web services among which, in the context of 3D, there are the Web Feature Service (WFS) and the 3D Portrayal Service (3DPS).

2.2.2. Web 3D Service (W3DS) and Web View Service (WVS).

The demand of serving large scale 3D city models and spatial data reflects the need of hierarchical data structures for 3D data such as OGC I3S and OGC 3D Tiles. Although these formats can transmit arbitrary sized geospatial data, they are not interoperable with consuming and visualization on the client (Koukofikis et al., 2018). The OGC 3DPS standard (Hagedorn et al., 2017) has been designed to enable the interoperable visualizations between various data providers and different browser-based 3D globes and other viewer implementations (Gutbell et al., 2016). An initial attempt to provide a solution regarding the interoperable 3D geovisualization was the following services: Web 3D Service (W3DS) and Web View Service (WVS).

2.2.2.1. Web 3D Service (W3DS)

Two versions of W3DS were published as OGC discussion papers (Quadt & Kolbe, 2005; Schilling & Kolbe, 2010). The Web 3D Service is a portrayal service for 3D geodata, delivering graphical elements from a given geographical area producing 3D scene graphs. These scene graphs are rendered by the client and can interactively be explored by the user (Figure 2-8).

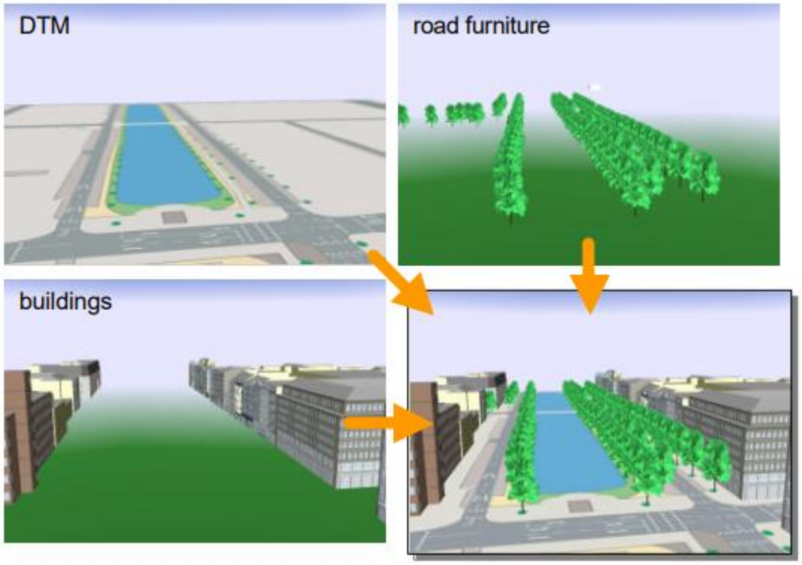


Figure 2-8: Different types of geodata are merged in one 3D scene graph using W3DS (Quadt & Kolbe, 2005)

The aforementioned service was implemented by Prieto et al., (2012) to achieve the visualization of CityGML file without plugins. The output format was X3D and the integration into web was achieved through X3DOM (Figure 2-9).

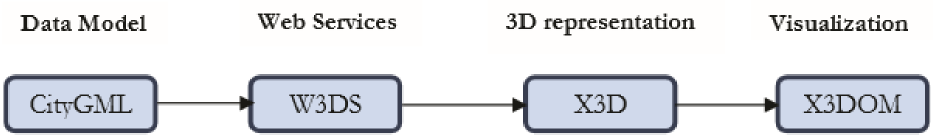


Figure 2-9: Use of W3DS for CityGML visualization and retrieval

(Prieto et al., 2012)

It should be noted that a basic design consideration for any client-server system is how to partition the workload between the client and the server. Depending on how the workload is distributed, WebGIS applications can be categorized as either thin client architecture or thick client architecture (Gong, 1999). However, the development of technologies used on both the server and client side led to the need to create an intermediate architecture, the medium client. According to previous architectures, W3DS belong to the medium client (Figure 2-10).

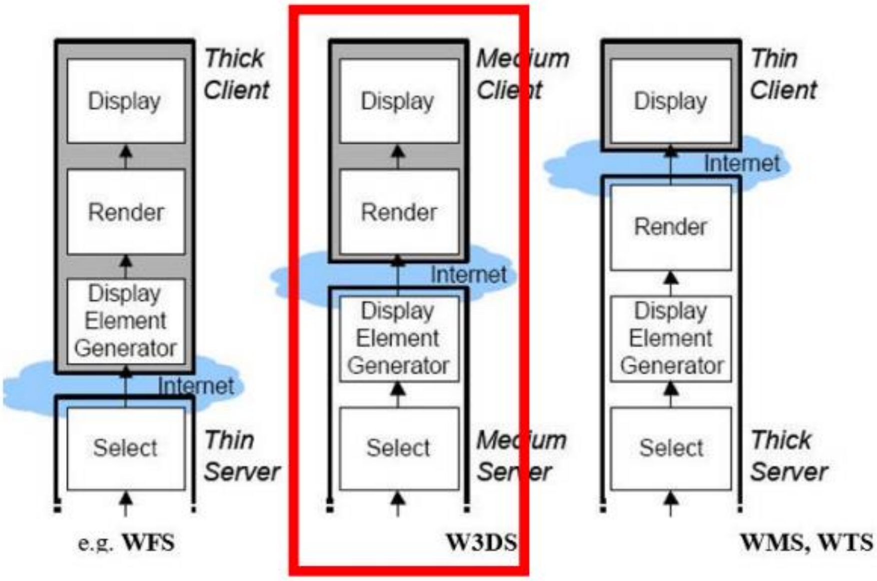


Figure 2-10: Medium Client Architecture

(Quadt & Kolbe, 2005)

2.2.2.2. Web View Service (WVS)

An alternative solution for retrieving and visualizing 3D data is the WVS. This service mainly provides 2D image representing a 3D view on a scene constructed from 3D geodata that is integrated and visualized by the WVS server (Hagedorn, 2010) (Figure 2-11). Additionally, WVS adopts the thin client architecture for visualizing, analyzing, navigating and retrieving 3D scene information. Consequently, the server should be equipped with the appropriate software and powerful graphics card and, from client's point of view, users could access to potentially complex 3D geodata with high-quality output and without having to provide and maintain specific 3D graphics hardware and software or streaming complex 3D data, since only standard images are transferred.

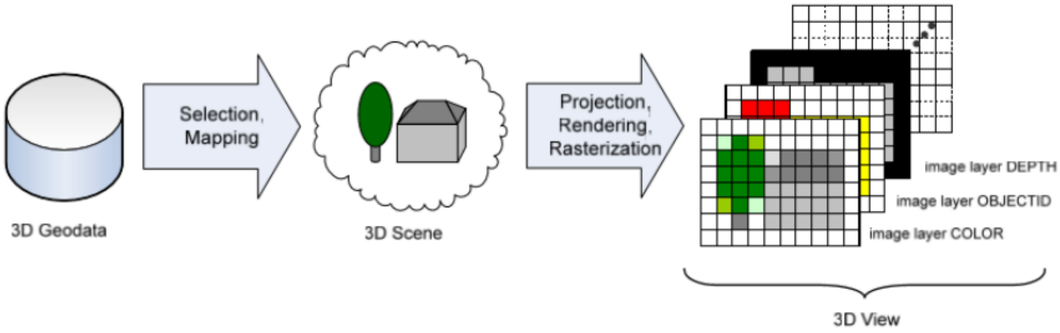


Figure 2-11: Retrieval and visualization of 3D data using WVS

(Hagedorn, 2010)

2.2.2.3. 3D Portrayal Interoperability Experiments (3DPIE)

In 2012, several experiments were presented by 3D Portrayal Interoperability Experiment (3DPIE) utilizing the 3D portrayal services W3DS and WVS (Schilling et al., 2012). The summary of these experiments including available data, servers, supported 3D portrayal services and exported formats are shown in Figure 2-12.

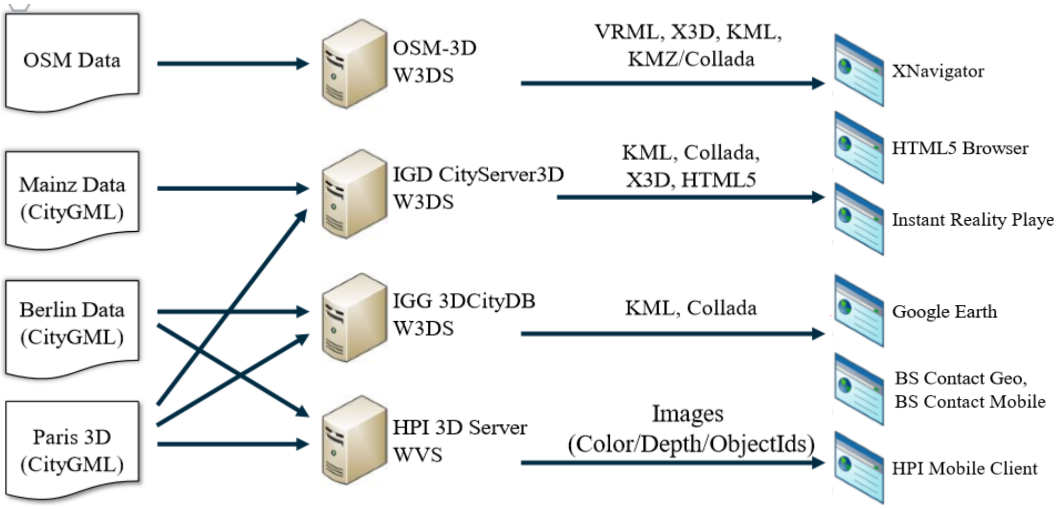


Figure 2-12: Experiments of 3DPIE

(Schilling et al., 2012) & modified by author

Totally, five service implementations of at least one of both standards, together with five clients were subjected to the 3DPIE. It emerged that several interoperability scenarios combining both approaches were possible, and that the differences between W3DS and WVS were significant but mostly reconcilable. However, some weaknesses also emerged. For instance, the problem of scaling to bigger geodata was tackled with the well-known tiling technique. Tiling does not easily translate to geometric 3D data, and thus, there is no one-size-fits-all solution. Despite this, the proposals put forward a limited but complex solution.

2.2.3. OGC 3D Portrayal Service (3DPS)

The 3DPS combines the essential parts of the proposed W3DS and WVS into a common interface and thus, it could provide either 3D graphics data or rendered images (Hagedorn et al., 2017). Consequently, it supports two fundamental 3D portrayal schemes and associated client/server configurations. The first one was implemented by Gaillard et al. (2015). They proposed a framework to visualize 3D city data stored natively in CityGML files. These files were cut into tiles with fixed size and thereafter, they were converted and stored on the server in JSON format, keeping any semantic information that could be stored in city objects. The retrieval of this data was achieved using the 3DPS GetScene request. On the other hand, Gutbell et al. (2016) implemented a server-side rendering framework to visualize 3D city models utilizing the 3DPS GetView operation.

Therefore, 3DPS interface specifies several operations that may be invoked by a 3DPS client and may be performed by a 3DPS service (Table 2-1).

OPERATIONS	DESCRIPTIONS
GetCapabilities	This operation allows a client to request information about a 3DPS server's capabilities and scene information offered
AbstractGetPortrayal	This is the abstract operation that forms the basis of the 3DPS operations *GetScene* and *GetView* and provide common parameters
GetResourceById	This operation allows a client to request arbitrary resource, as indicated by the service
GetScene	This operation allows a client to retrieve a 3D scene represented as 3D geometries and texture data, organized as a scene graph and/or spatial index
GetView	This operation allows a client to retrieve a 3D view of a scene represented as image
AbstractGetFeatureInfo	This is the abstract operation that forms the basis for specific getFeatureInfo operations that allow a client to retrieve more information about portrayed features
GetFeatureInfoByRay	This operation allows a client to retrieve information about features that are selected based on a virtual ray
GetFeatureInfoByPosition	This operation allows a client to retrieve information about features that are selected based on location
GetFeatureInfoByObjectId	This operation allows a client to retrieve information about features that are selected based on object identifiers

Table 2-1: Operations of 3DPS

In 2018, several experiments were presented by the OGC testbed 13 Engineering Report documents (Coors, 2018). The main goal of this report was to test and validate the interoperability of the OGC 3DPS, using 3DPS implementation instances to generate web-based visualizations with a workflow that used CityGML as data sources and 3D Tiles and I3S as data delivery formats. For this purpose, several processing algorithms were developed to convert CityGML into either 3D Tiles or I3S delivery Formats. More specific, Analytical Graphics Inc (AGI) created the necessary processing algorithms to convert CityGML into 3D Tiles within its 3D Tiles processing Tools; ESRI provided processing algorithms to convert CityGML into I3S within ArcGIS and by using FME (Feature Manipulation Engine); Fraunhofer and the SME virtualcitySYSTEMS created processing algorithms to convert CityGML with and without Application Domain Extension (ADE) into 3D Tiles as extension on top of GeoRocket[9]. As a result, this report summarizes a proof-of-concept of the use of 3D Tiles and I3S as data delivery formats for the OGC 3DPS interface standard (Koukofikis et al., 2018).

However, although the interoperable portrayal of the 3D city models has been achieved, this requires complex processing algorithms to convert CityGML into the appropriate OGC portrayal standards such as I3S and 3D Tiles. Consequently, the utilization of 3D Web Portrayal Services is not the optimal solution for the current thesis objective. Therefore, the interoperable and easy-to-use information retrieval of a CityGML based on its semantic characteristics will be further examined using the OGC Web services for sharing and managing raw data such as WFS (see section 2.3).

2.3. OGC Web Services for Sharing and Managing Raw Data

The OGC Web service Standard for reading and writing geographic features in vector format is the WFS. With WFS, clients can perform operations, including insert, update, delete and query for geospatial data residing on the server (Vretanos, 2010). Therefore, this international standard provides a standardized and open interface for requesting

[9] **GeoRocket** is a high-performance data store for geospatial files such as 3D city models (CityGML), GML and GeoJSON files.

geographic features across the web using platform-independent calls and thus, it allows clients to only retrieve or modify the data they are seeking rather than retrieving a file that contains possibly much more. More specific, the OGC WFS 2.0 interface define eleven operations that can be invoked by a client (Table 2-2). However, a WFS server is not required to offer all these operations to conform to the standard but may support a subset only. Hence, the WFS standard defines conformance classes such as simple WFS, Basic WFS, Transactional WFS and Locking WFS that grow in the number of mandatory operations.

OPERATIONS	DESCRIPTIONS
GetCapabilities	The GetCapabilities operation generates a service metadata document describing the WFS services provided by as server.
DescribeFeatureType	The DescribeFeatureType operation requests the structure of the feature type that WFS support.
ListStoredQueries	The ListStoredQueries operation lists the stored queries available at the server.
DescribeStoredQuery	The DescribeStoredQueries operation provides detailed metadata about each stored query expression that server offers.
GetFeature	The GetFeature operation retrieves a geographic feature and its attributes to match a filter query.
GetPropertyValue	The GetPropertyValue operation allows the value of a feature property or part of the value of a complex feature property to be retrieved from the data store for a set of features identified using a query expression.
LockFeature	The LockFeature operation requests the server to lock on one or more features for the duration of the transaction such as update or delete.
GetFeatureWithLock	The GetFeatureWithLock operation is functionally similar to the GetFeature operation except that in response to a

	GetFeatureWithLock operation, WFS shall not only generate a response document similar to that of the GetFeature operation but shall also lock the features in the result set.
CreateStoreQuery	The CreateStoreQuery operation is used to create a stored query
DropStoredQuery	The DropStoredQuery operation allows previously created stored queries to bed dropped from the system.
Transaction	The Transaction operation requests the server to create, update and delete geographic features

Table 2-2: Supported WFS 2.0 operations

(Vretanos, 2010)

2.3.1. Extending the OGC WFS 2.0 standard

Retrieving CityGML data via a OGC WFS 2.0 and previous versions 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. An instance of this complexity regarding the building module of the CityGML is presented in 3DCityDB (3D City Database)[10] logical design overview in Figure 2-13.

CityGML as an information model and GML application schema makes extensive use of complex data types for properties and nesting of features within feature collections. Consequently, CityGML can contain very deeply nested data structures. Additionally, the range of geometry types used in CityGML are not fully supported by relational databases, addressing several issues for implementing WFS on top of them. Consequently, a variety of research was conducted on the extension of the OGC WFS.

[10] **3D City Database** is a free Open Source package consisting of a database schema and a set of software tools to import, manage, analyze, visualize, and export virtual 3D city model according to the CityGML standard.

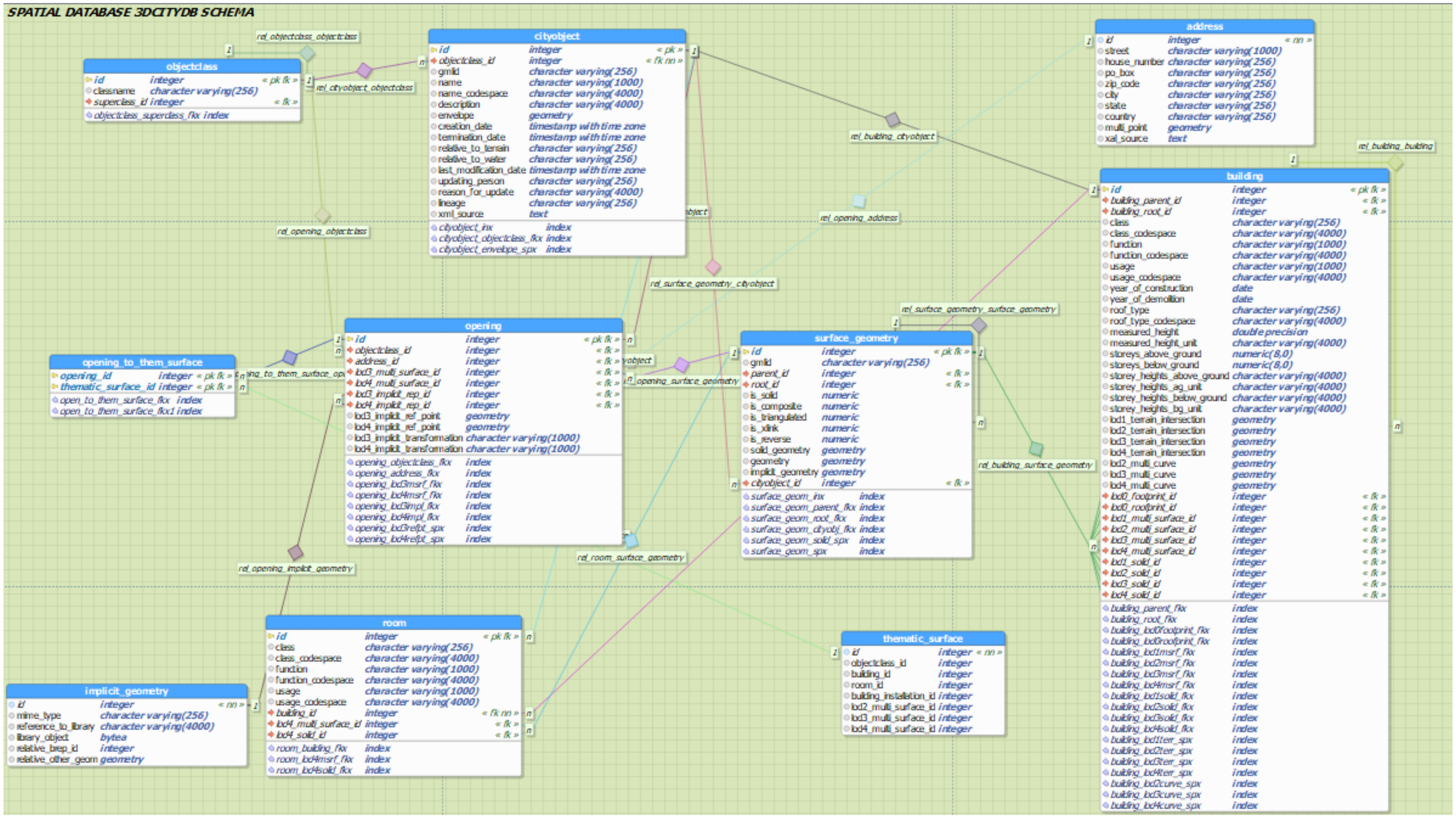

SPATIAL DATABASE 3DCITYDB SCHEMA
objectclass
id integer « pk fk »
classname character varying(256)
superclass_id integer « fk »
objectclass_superclass_fkx index
rel_objectclass_objectclass
rel_cityobject_objectclass
cityobject
id integer « pk »
objectclass_id integer « fk nn »
gmlid character varying(256)
name character varying(1000)
name_codespace character varying(4000)
description character varying(4000)
envelope geometry
creation_date timestamp with time zone
termination_date timestamp with time zone
relative_to_terrain character varying(256)
relative_to_water character varying(256)
last_modification_date timestamp with time zone
updating_person character varying(256)
reason_for_update character varying(4000)
lineage character varying(256)
xml_source text
cityobject_inx index
cityobject_objectclass_fkx index
cityobject_envelope_spx index
address
id integer « nn »
street character varying(1000)
house_number character varying(256)
po_box character varying(256)
zip_code character varying(256)
city character varying(256)
state character varying(256)
country character varying(256)
multi_point geometry
xal_source text
rel_building_cityobject
rel_opening_address
rel_building_building
rel_opening_objectclass
rel_surface_geometry_cityobject
rel_surface_geometry_surface_geometry
building
id integer « pk fk »
building_parent_id integer « fk »
building_root_id integer « fk »
class character varying(256)
class_codespace character varying(4000)
function character varying(1000)
function_codespace character varying(4000)
usage character varying(1000)
usage_codespace character varying(4000)
year_of_construction date
year_of_demolition date
roof_type character varying(256)
roof_type_codespace character varying(4000)
measured_height double precision
measured_height_unit character varying(4000)
storeys_above_ground numeric(8,0)
storeys_below_ground numeric(8,0)
storey_heights_above_ground character varying(4000)
storey_heights_ag_unit character varying(4000)
storey_heights_below_ground character varying(4000)
storey_heights_bg_unit character varying(4000)
lod1_terrain_intersection geometry
lod2_terrain_intersection geometry
lod3_terrain_intersection geometry
lod4_terrain_intersection geometry
lod2_multi_curve geometry
lod3_multi_curve geometry
lod4_multi_curve geometry
lod0_footprint_id integer « fk »
lod0_roofprint_id integer « fk »
lod1_multi_surface_id integer « fk »
lod2_multi_surface_id integer « fk »
lod3_multi_surface_id integer « fk »
lod4_multi_surface_id integer « fk »
lod1_solid_id integer « fk »
lod2_solid_id integer « fk »
lod3_solid_id integer « fk »
lod4_solid_id integer « fk »
building_parent_fkx index
building_root_fkx index
building_lod0footprint_fkx index
building_lod0roofprint_fkx index
building_lod1msrf_fkx index
building_lod2msrf_fkx index
building_lod3msrf_fkx index
building_lod4msrf_fkx index
building_lod1solid_fkx index
building_lod2solid_fkx index
building_lod3solid_fkx index
building_lod4solid_fkx index
building_lod1terr_spx index
building_lod2terr_spx index
building_lod3terr_spx index
building_lod4terr_spx index
building_lod2curve_spx index
building_lod3curve_spx index
building_lod4curve_spx index
opening_to_them_surface
opening_id integer « pk fk »
thematic_surface_id integer « pk fk »
open_to_them_surface_fkx index
open_to_them_surface_fkx1 index
opening
id integer « pk fk »
objectclass_id integer « fk »
address_id integer « fk »
lod3_multi_surface_id integer « fk »
lod4_multi_surface_id integer « fk »
lod3_implicit_rep_id integer « fk »
lod4_implicit_rep_id integer « fk »
lod3_implicit_ref_point geometry
lod4_implicit_ref_point geometry
lod3_implicit_transformation character varying(1000)
lod4_implicit_transformation character varying(1000)
opening_objectclass_fkx index
opening_address_fkx index
opening_lod3msrf_fkx index
opening_lod4msrf_fkx index
opening_lod3impl_fkx index
opening_lod4impl_fkx index
opening_lod3refpt_spx index
opening_lod4refpt_spx index
rel_opening_surface_geometry
surface_geometry
id integer « pk fk »
gmlid character varying(256)
parent_id integer « fk »
root_id integer « fk »
is_solid numeric
is_composite numeric
is_triangulated numeric
is_xlink numeric
is_reverse numeric
solid_geometry geometry
geometry geometry
implicit_geometry geometry
cityobject_id integer « fk »
surface_geom_inx index
surface_geom_parent_fkx index
surface_geom_root_fkx index
surface_geom_cityobj_fkx index
surface_geom_solid_spx index
surface_geom_spx index
rel_building_surface_geometry
rel_opening_implicit_geometry
rel_room_surface_geometry
room
id integer « pk fk »
class character varying(256)
class_codespace character varying(4000)
function character varying(1000)
function_codespace character varying(4000)
usage character varying(1000)
usage_codespace character varying(4000)
building_id integer « fk nn »
lod4_multi_surface_id integer « fk »
lod4_solid_id integer « fk »
room_building_fkx index
room_lod4msrf_fkx index
room_lod4solid_fkx index
implicit_geometry
id integer « nn »
mime_type character varying(256)
reference_to_library character varying(4000)
library_object bytea
relative_brep_id integer
relative_other_geom geometry
thematic_surface
id integer « nn »
objectclass_id integer
building_id integer
room_id integer
building_installation_id integer
lod2_multi_surface_id integer
lod3_multi_surface_id integer
lod4_multi_surface_id integer
1
n

Figure 2-13: Logical design of 3DcityDB database regarding building module of CityGML

(Athanasiou et al., 2018)

2.3.1.1. Snowflake CityGML WFS

In 2006, the OGC Web Service-Phase 4(OWS-4) testbed was taken place under the initiative of OGC's Interoperability Program to collaboratively extend and demonstrate OGC's baseline for geospatial interoperability. In the context of this testbed the serving of CityGML via WFS was addressed (Curtis, 2008). As a result, the Snowflake CityGML WFS was created allowing basic operations of WFS specification such as DescribeFeatureType, GetCapabilities and GetFeature.

Additionally, it supports the following features of CityGML: Building, CityObjectGroup, GenericCityObject, ReliefFeature and CityFurniture in all LoDs. The mechanism regarding the data request and response using the Snowflake CityGML WFS is presented in Figure 2-14.

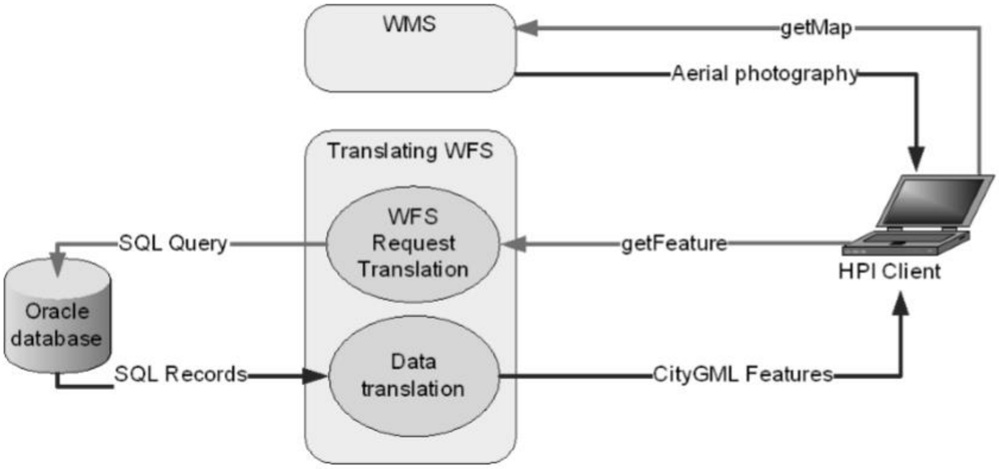


Figure 2-14: Data request and response using the Snowflake CityGML WFS

(Curtis, 2008)

According to this methodology, the CityGML data should be stored into an Oracle database using a GML bulk loading tool called GO Loader. Thereafter, the data request and response among client and database is achieved using a data translation engine called GO publisher.

2.3.1.2. 3DCityDB WFS

Within the same research context, Yao et al. (2018) implemented the OGC WFS 2.0 in conjunction with 3D City Database, which supports multiscale and rich semantic structure of CityGML and developed the 3DCityDB WFS (Figure 2-15). When sending a request to the 3DCityDB WFS server to retrieve certain CityGML Features, the 3DCityDB WFS servlet captures and parses this request and translate it to a corresponding database query to obtain a list of the respective GMLIDs of the CityGML top-level features. Thereafter, these feature IDs will be handed over to the CityGML Import/Export module which utilizes its pre-complied citygml4j/JAXB classes as well as the multi-threading API for efficiently querying and generating the corresponding CityGML XML elements. Finally, these XML datasets will be returned as a response of the WFS request. The Open Source version of the 3DCityDB WFS implements the Simple WFS conformance classes and therefore, it only handles GetFeatureById queries which is enough to retrieve objects by their GMLID. However, ad-hoc queries or semantic retrieval of available features are not supported.

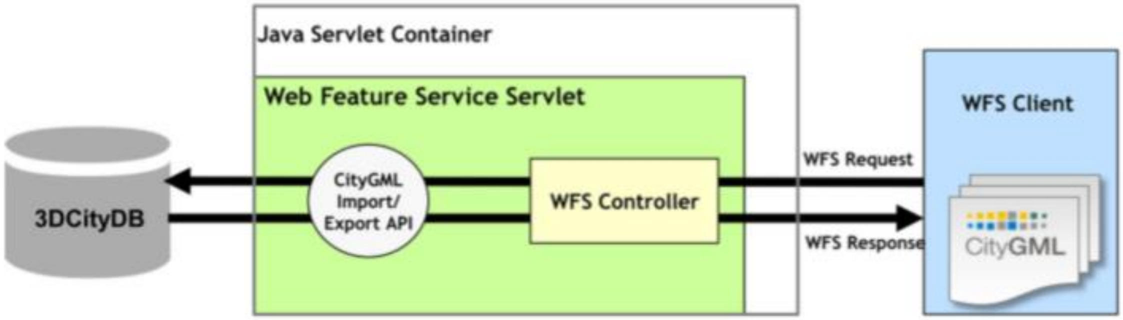


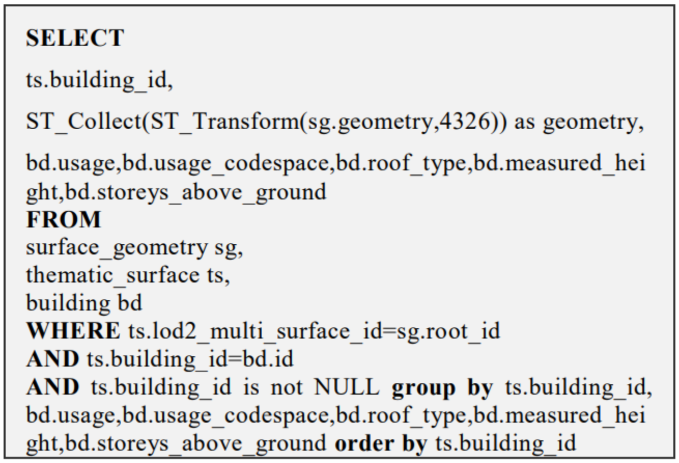
Figure 2-15: 3DCityDB WFS

(Yao et al., 2018)

2.3.1.3. Extending the WFS of GeoServer

Another approach was that of Zhu, et al. (2016), who focused on the open source solution to serve CityGML data via a WFS with advanced functionality. Therefore, the GeoServer[11] was tested in combination with its Application Schema extension since it supports the OGC WFS 2.0 standard and provides full-fledged WFS functionality including discovery, query, locking, transaction and stored query operations. The complex feature types of CityGML could be mapped by GeoServer Application Schema using its two available concepts such as Feature Mapping and Feature Chaining. There are some limitations to this approach and the most important is that all public GML application schemas used for mapping with GeoServer Application Schema must meet the GML encoding rules. However, not all schemas of CityGML obey these rules.

Similarly, a GeoServer approach was implemented by Pispidikis et al. (2016) for the visualization of CityGML data via the WFS 2.0 standard. Therefore, a PostGIS database was used in compliance with 3DCityDB schema and connected to GeoServer. Then, a suitable view was created by the use of SQL query shown in Figure 2-16.



```
SELECT
ts.building_id,
ST_Collect(ST_Transform(sg.geometry,4326)) as geometry,
bd.usage,bd.usage_codespace,bd.roof_type,bd.measured_hei
ght,bd.storeys_above_ground
FROM
surface_geometry sg,
thematic_surface ts,
building bd
WHERE ts.lod2_multi_surface_id=sg.root_id
AND ts.building_id=bd.id
AND ts.building_id is not NULL group by ts.building_id,
bd.usage,bd.usage_codespace,bd.roof_type,bd.measured_hei
ght,bd.storeys_above_ground order by ts.building_id
```

Figure 2-16: SQL query for creating the Lod2 building view of CityGML

(Pispidikis et al., 2016)

[11] **GeoServer** is a java-based software server that allows users to view and edit and share geospatial data.

2.3.2. Making the OGC WFS RESTful

Extending WFS to support the retrieval of CityGML data is considered very important. However, the WFS 2.0 and previous versions used a Remote-Procedure-Call-Over-HTTP architectural style implementing XML for any payload. This architecture was considered state-of-the-art when the WFS standard was originally designed in the late 1990s and early 2000s (Portele & Vretanos, 2018). However, the evolution of the Web 2.0 phenomenon has led to the increased adoption of the RESTful Service paradigm which takes full advantage of the web technology, making correct use of the HTTP protocol and also follows the Resource-Oriented Architecture (ROA) (Pispidikis & Dimopoulou, 2018). Additionally, REST as a different approach to provide access to data, it can be used to provide end users with a guided, prepackaged way of accessing data or resources. On the other hand, WFS, as a query language, enables end users to submit any type of supported WFS request. Consequently, due to the limitless nature of WFS, difficulties in query optimization can arise. Therefore, REST can be utilized to steer the end user towards a predefined pattern of access such as tiles, collections and IDs.

2.3.2.1. GO publisher RESTful service

The Snowflake Software (2016) presented the GO publisher RESTful service as a simple web interface which allows HyperText Transfer Protocol (HTTP) request to be converted and redirected to a GO Publisher WFS which provides access to XML/GML resources. As a result, this RESTful service works on top of the respective WFS providing specific Uniform Resource Locator (URL) resources to the end users (Figure 2-17).

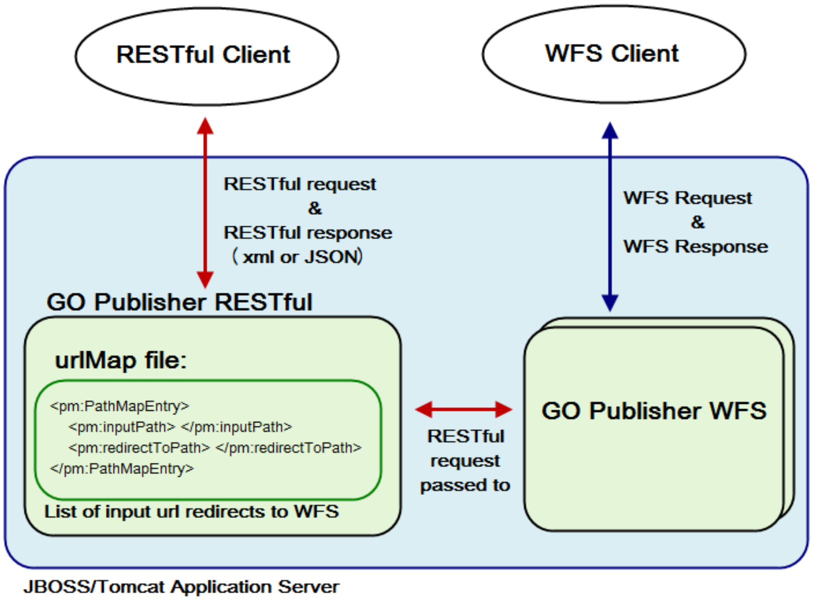

RESTful Client
WFS Client
RESTful request & RESTful response (xml or JSON)
WFS Request & WFS Response
GO Publisher RESTful
urlMap file:
<pm:PathMapEntry>
<pm:inputPath> </pm:inputPath>
<pm:redirectToPath> </pm:redirectToPath>
</pm:PathMapEntry>
List of input url redirects to WFS
RESTful request passed to
GO Publisher WFS
JBOSS/Tomcat Application Server

Figure 2-17: Image of GO Publisher RESTful service working with GO Publisher WFS (Snowflake Software, 2016).

2.3.2.2. OGC API-Features

This REST-based architecture was adopted by the version 3 of WFS (Portele & Vretanos, 2018), now called OGC API-Features (Portele, Vretanos & Heazel, 2019). Therefore, this version of the WFS standard uses a resource architecture and specifies a RESTful service interface providing resources regarding features and feature collection respectively. So, the list of a feature collection (e.g. buildings) can be retrieved using the following request:

```
../collections/buildings/items
```

Thereafter, each feature in a feature collection can also be requested by implementing the respective id as sub-resource.

```
../collections/buildings/items/{id}
```

However, the core of the OGC API Features does not currently support the implementation of additional sub-resources so that the semantic retrieval of CityGML Data is fully achieved but provides a solution to this limitation by extending the Core API including richer queries from existing OGC standards (Portele, 2019). The said solution is quite sufficient, since the OGC API-Features is intended to provide a general solution for retrieving data regarding all available standards of the OGC API family. However, this implies and requires good knowledge for both the structure of the source (e.g. CityGML) and the respective syntax of the implemented OGC standard such as OGC Filter Encoding Standard 2.0, OGC Common Query Language (CQL) or other query languages or data platforms such as Falcor and GraphQL (Portele, 2019).

CHAPTER 2: RETRIEVING CITYGML INFORMATION

3. CITYGML RESTFUL WEB SERVICE

This chapter examines the use of non-OGC Web services for the interoperable and easy-to-use information retrieval of a CityGML based on its semantic characteristics. For this purpose, The SOAP and REST Web services are further studied and compared. Also, the REST WS are compared with new state-of-the-art technologies that can be adopted as a CityGML data retrieval mechanism such as GraphQL and Falcor. Thereafter, several principles and guideline are addressed with regard to the CityGML RESTful Web service and finally, the conceptual design of the "CityModels" and "Gmlid" resources is presented.

In this chapter, the 3rd sub-research question of the current dissertation is answered:

> *What is the most appropriate architecture type of a web service for achieving the easy-to-use information retrieval of a city?*

This chapter utilizes the following papers:

(1) **Pispidikis** and Dimopoulou (2018)
(2) **Pispidikis** and Dimopoulou (2019)
(3) Chatzinikolaou, **Pispidikis** and Dimopoulou (2020)

3.1. The Solution of REST Approach

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 (Gröger et al., 2012). Therefore, the architecture of the CityGML 2.0 is shaped via five key components (Figure 3-1).

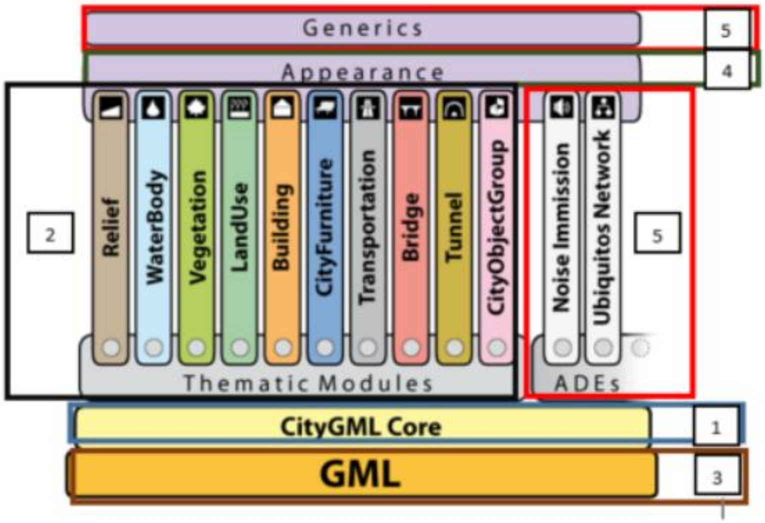


Figure 3-1: CityGML Architecture

(Pispidikis & Dimopoulou, 2018).

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. 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 Geography Markup Language 3 (GML 3). The fourth component contains the 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 Extension) modules (Gröger et al., 2012).

Taking into account the CityGML architecture (Figure 3-1), it is concluded that its structure is rather semantic than geometric and therefore, the retrieval of the data has to be achieved mainly in compliance with the CityGML's semantic information. On the other hand, the OGC WFS is a geospatial Web service, which means that it was developed with the aim of retrieving, visualizing and modifying data based on geometry. Consequently, the interoperable and easy-to-use information retrieval of CityGML based on its semantic characteristics will be further examined using non- OGC Web services by focusing on different interoperable approaches.

3.1.1. SOAP Vs REST

The communication between a Web service and a client involves the client sending requests to the Web service, and the Web service response request to the client. Depending on the format of communication used, there are two types of Web services such as the SOAP-based Web services and the REST-style Web services.

SOAP (Simple Object Access Protocol) is a protocol specification for exchanging structured information in XML format. Specifically, a SOAP messages packages an XML body in an XML envelope and the respective request is send via HTTP POST method. Because of this structured, SOAP is difficult to be constructed and parsed manually. Fortunately, the solution of the latter is achieved by the implementation of a variate of tools in conjunction with WSDL (Web services Description Language)(Fu & Sun, 2010).

REST (Representational State Transfer) which is a result of the Roy Fielding's dissertation (Fielding & Taylor, 2000) is a style of software architecture which is designed to fully take advantages of HTTP, while reducing system complexity and improving system scalability (Richardson & Ruby, 2007). Moreover, this architecture was implemented to avoid the use of complex data exchange mechanism such as COBRA (Common Object Request Broker Architecture), RPC (Remote Procedure Call) or SOAP. In the most common implementation of REST, all requests are made by a URL, and all parameters are in the URL. REST does not define standards for the server response format, but JSON and XML are frequently used without SOAP encapsulation (Fu & Sun, 2010).

REST has gained widespread acceptance across the Web as a more flexible 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). 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 & 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. 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 protocols concluded that SOAP is preferable for financial, banking, telecommunication services, and REST for Social interaction, Web chat, and mobile services. Tihomirovs & Grabis (2016) performed a comparison between SOAP and REST using software evaluation metrics and concluded that is not possible to clearly identify the best approach to ensure data exchange because each integration project should be assessed individually. However, they pointed out that if the project requires greater scalability, compatibility and performance, it is better to choose REST. On the other hand, the SOAP is a better choice when a project requires security and reliability, easier maintainability on the client side, as well as a lower number of possible errors.

In conclusion, 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.

3.1.2. GraphQL and Falcor

3.1.2.1. GraphQL

In 2016, Facebook released a specification and a reference implementation of the GraphQL framework. This framework introduces a new type of Web-based data access interfaces that presents an alternative to the notion of REST-based interfaces (Hartig & Pérez, 2018). GraphQL is a query language for APIs and a runtime for fulfilling those queries with the existing data. It provides a complete and understandable description of the data in available API, gives clients the power to ask for exactly what they need and nothing more, makes it easier to evolve APIs over time, and enables powerful developer tools. It was developed to address the need for more flexibility and efficiency solving many of the shortcomings and inefficiencies that developers experience when interacting with REST APIs (GraphQL is the better REST, n.d.). REST encourages versatile resource-oriented architecture where self-contained cohesive resources are identified by URLs and are accessed or manipulated via multiple HTTP endpoints (Vogel et al., 2017). The most common problem with this approach is that of overfetching. Overfetching means that the clients download more information than in actually required in the app, as they are limited to perform predefined operations that may have been designed by API providers regardless of the clients' specific requirements (Wittern et al., 2018). For instance, when a client needs to display a list of buildings only with their respective function attribute, then, in a REST API, this resource would have the following endpoint:

```
/buildings
```

The result of this request will be a JSON array with building data which may contain more information about the buildings e.g. usage, class etc. which is useless for the client.

Another issue is the underfetching and the n+1 request problem. Generally, underfetching means that a specific endpoint does not provide enough of the required information and hence, multiple endpoints should be requested. For example, when a client needs information about a specific room of a building then more than one endpoint should be requested (Figure 3-2).

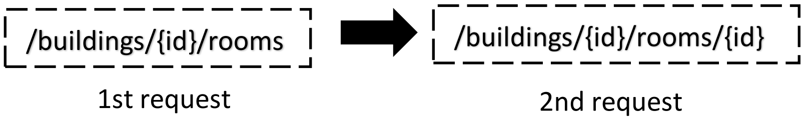
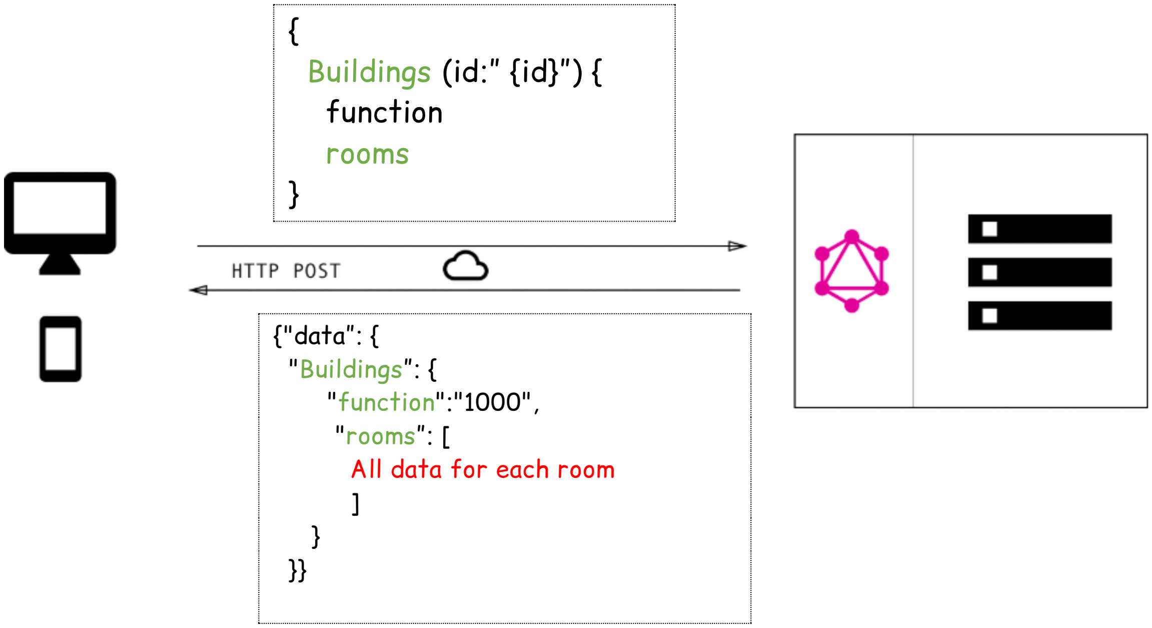


Figure 3-2: Example of underfetching problem using REST-based request

On the other hand, GraphQL promotes a more data-centric model without architecture resources. A GraphQL service represents an object graph of data entities which are collectively accessible through a single endpoint and URL. Therefore, the GraphQL's solution to the aforementioned issues is a query language that allows clients to specify exact data requirements on a data field level, executing the desirable request using only one endpoint. The solution of the above-mentioned examples using the GraphQL is shown in Figure 3-3. It should be noted that a client could semantically retrieve CityGML data when the corresponding query of GraphQL request is suitably structured.



```
{
  Buildings (id:" {id}") {
    function
    rooms
}
```

```
{"data": {
  "Buildings": {
      "function":"1000",
       "rooms": [
         All data for each room
         ]
    }
  }}
```

Figure 3-3: Example of GraphQL request

However, although GraphQL is considered a promising candidate for being used as a data retrieval mechanism regarding CityGML, there are several issues to be addressed. The main issue arises from the fact that the users need to have advanced knowledge for both the structure of the GraphQL query language and the source. Additionally, according to Portele (2019), there is no support for geometries or spatial queries in GraphQL.

3.1.2.2. Falcor

Similar to GraphQL, Falcor, as data platform that powers the Netflix user interfaces (Falcor, n.d.), was designed to solve the same problem that focuses on managing the increasingly complex data requirements of modern web and mobile apps (Helfer, 2016). Falcor provides an alternative solution to retrieve data having the starting point that all data is a single virtual JSON object (Figure 3-4) and the data retrieval is achieved in a same way whether the data is on the client, or on the server. This allows clients to work with data using standard paths and operations such as get, set and call. Using Falcor, the over and under fetching are not an issue since the clients can retrieve the desirable data according to their needs. However, Falcor has no schema of the data and assumes the data is known (Falcor, n.d.). Additionally, it is mainly designed for use in JavaScript and thus, it has no support for geometries or spatial predicates (Portele, 2019)

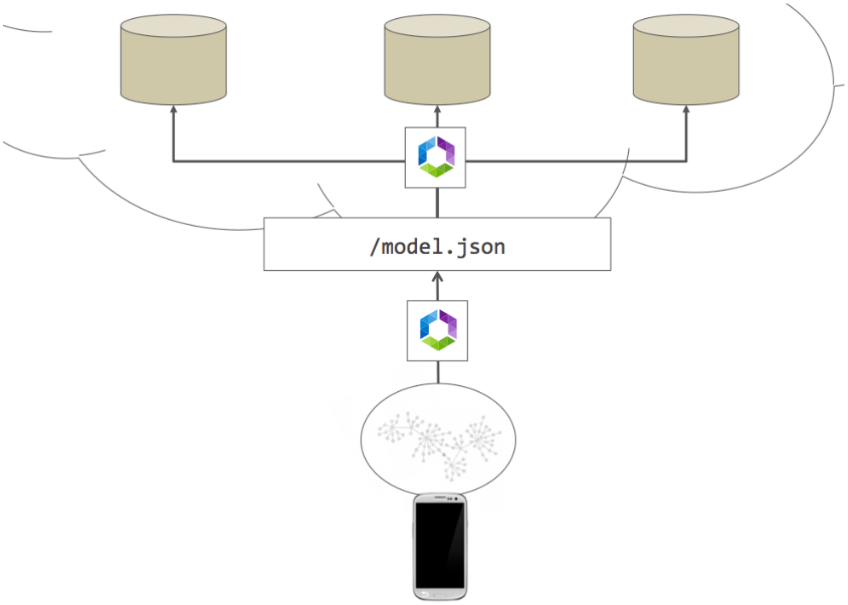


Figure 3-4: JSON-based data retrieval using Falcor

(Falcor, n.d.)

3.1.2.3. Results

The current section presents state-of-the-art technologies that can be adopted as a CityGML data retrieval mechanism. According to the core research question of the current thesis (see section 1.3) the CityGML data retrieval should be achieved in compliance with the following keywords: interoperability, easy-to-use, semantically and non-expert user. The implementation of the Falcor or GraphQL presupposes that the client should have good knowledge of either the GraphQL query language or the complex CityGML schema. Additionally, taking into consideration the complexity of the CityGML and the fact that the CityGML data needs to be semantically retrieved, the ROA architecture should be adopted. As a result, the REST-based architecture style is chosen and the CityGML RESTful Web service is conceptually designed.

3.1.3. Principles of RESTful Web services

The evolution of the Web 2.0[12] phenomenon has led to the increase adoption of the RESTful services paradigm (Lathem et al., 2007). RESTful Web services work on the web, taking full advantages and making correct use of the HTTP protocol (Webber et al., 2010). As a protocol, HTTP defines a set of rules and procedures that both Web clients and Web servers used to communicate with each other (Fun & Su, 2010). Therefore, via HTTP, a Web server knows what information to put in the message header and body, and the Web client knows what to expect from the response header and body respectively. RESTful Web services follow the ROA architecture and hence, everything that a service provides has to be a resource. Resources are identified by URIs (Uniform Resource Identifier), which provide a global addressing space for resource and service discovery (Rodriguez, 2008).

3.1.3.1. Constraints of the REST architecture style

The main design constraints of the REST architecture style can be summarized as follows:

- 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 and thus, a URI identifies only one resource, but a resource can have more than one URIs.

- Uniform Interface: all resources are managed via a uniform interface. In HTTP, the uniform interface comprises a variety of methods of request such as GET, POST, HEAD, PUT and DELETE that can be applied to all identifiers of Web resources. Each of these methods should be used for specific operations such as create, read, update and delete (CRUD). More specific, PUT updates a resource, which can be deleted using DELETE method. GET is used to retrieve the current state of resource in some representation

[12] **Web 2.0** does not refer to any technical upgrades to the internet, rather, it simply refers to a shift in how it is used. It describes the new age of internet – a higher level of information sharing and interconnectedness among participants. Web 2.0 allows users to actively participate in the experience and not just act as passive viewer who intake information.

and POST is used to insert a new resource. It should be noted that GET, PUT and DELETE are characterized as idempotent[13] methods since they can be safely repeated, while POST is non-idempotent method (Table 3-1).

METHODS	OPERATIONS	RIGHTS	
POST	**C**reate	Read	Non-idempotent
GET	**R**ead (Retrieve)	Write	Idempotent
PUT	**U**pdate (Modify)	Write	Idempotent
DELETE	**D**elete	Write	Idempotent

Table 3-1: HTTP methods

- Statelessness: every HTTP request happens in complete isolation. Therefore, 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 that contain both the representation of resource, which can be accessed in a variety of formats such as XML and JSON 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. Therefore, it is reasonable to model state transitions between resources as metadata. Having a metadata model that describes the state transitions enables to exploit the model in order to apply access control. Thereafter, state transitions that must not performed by a client can be skipped and not included in the response. As a result, the unnecessary network traffic is reduced and security is increased (Somoza Alonso, 2017).

[13] **Idempotence** is the property of certain operations in mathematics and computer science that can be applied multiple times without changing the result beyond the initial application.

3.1.3.2. Richardson maturity model

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 3-5).

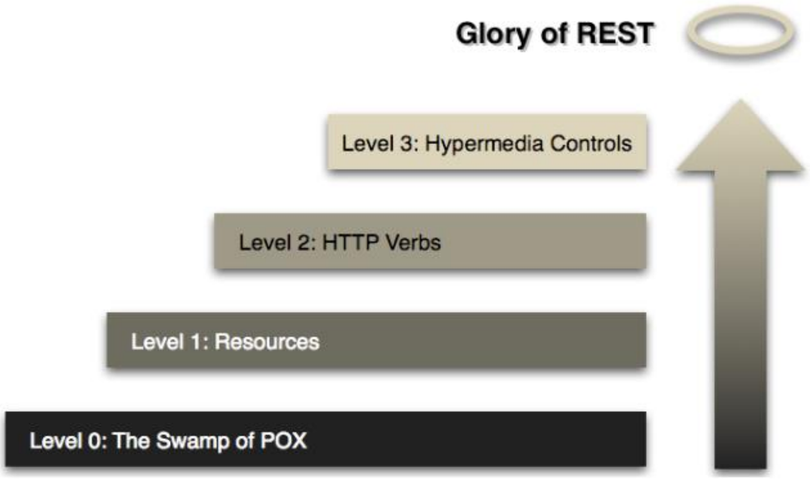


Figure 3-5: Richardson maturity model for RESTful Web services

(Fowler, 2010)

- Level 0: The system is distributed and invokes remote procedure calls without using any of the mechanisms of the Web. These might be some sort of reusable methods that offer specific services.

- Level 1 - Resources: Resource orientation is the most fundamental design guideline for REST. Instead of making all request to a singular endpoint, resources are targeted individually and therefore, each resources has a unique address.

- Level 2 – HTTP Verbs: HTTP verbs such as GET, POST, PUT and DELETE determine the action that is performed on resources instead of encapsulating the method into the resources address. Hence, the resource address only consist of nouns and the HTTP protocol carries the action.

- Level 3 – Hypermedia Controls: the system applies the HATEOAS constrain, which means that a server sends any possible state transitions with the resource to the client.

3.2. Methodology for the RESTful-based CityGML retrieval

Several Principles and guidelines should be adopted so that retrieving CityGML data is achieved by utilizing the RESTful-based architecture style. Therefore, the retrieval mechanism of CityGML RESTful Web service is structured in compliance with the ROA architecture and hence, everything that a service provides is a resource. The name of every resource is 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 methods and since the RESTful Web service is designed in compliance with the HTTP specification, the data is retrieved implementing the HTTP GET method. Additionally, the CityGML RESTful Web service is information-based and not Geometric-based Web service as the complex structure of CityGML is more semantic rather than geometric. The methodological steps for the conceptual design of the CityGML RESTful Web service initially include the configuration of the main resource schema based on the Resource Oriented Architecture as well as the definition of the main resources, the information retrieval and the filters that may need to be applied. Thereafter, the sub-resources should be defined so that the retrieval of all objects of a 3D city model could be achieved semantically. Finally, all of the aforementioned steps should be designed based on the constraints of the RESTful approach and thus, the CityGML RESTful Web service should guide the user in easy-to-use data retrieval.

3.2.1. Thematic resources

3.2.1.1. Main resources

Taking into consideration the five components of the CityGML's architecture (see Figure 18), only the second one (ten thematic modules) defines the semantic features of CityGML.

Therefore, these thematic modules should be the main resources of the CityGML RESTful Web service (Figure 3-6). The names of these main resources are based on the namespace prefix of CityGML v2 specification (Gröger et al., 2012) and they are shown in table 3-2.

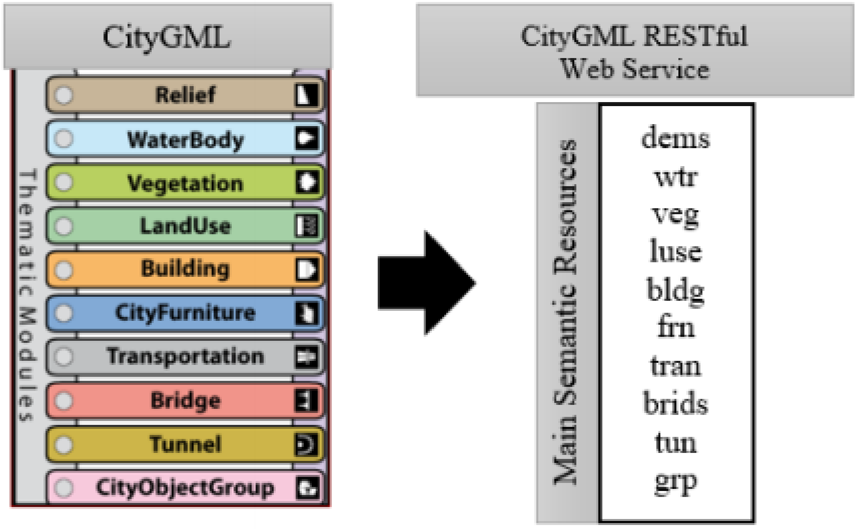


Figure 3-6: Main resources of CityGML RESTful Web service

(Pispidikis & Dimopoulou, 2019)

Resource Name	URI	CityGML Modules
bldg	../bldg	Building
wtr	../wtr	Waterbody
dems	../dems	Relief
veg	../veg	Vegetation
luse	../luse	LandUse
frn	../frn	CityFurniture
tran	../tran	Transportation
brid	../brid	Bridge
tun	../tun	Tunnel
grp	../grp	CityObjectGroup

Table 3-2: Name of the main resources according to the namespace prefix of CityGML v2

However, some extra main semantic resources are also defined to make it easier to access their available semantic features. These extra main resources are part of the main resources such as “tran” and “veg” (Figure 3-7).

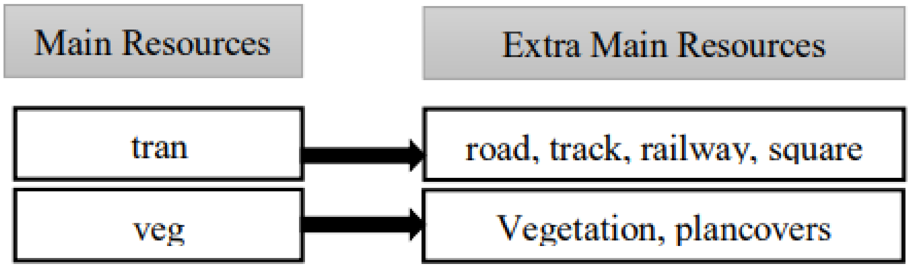


Figure 3-7: Extra main resources

(Pispidikis & Dimopoulou, 2019)

3.2.1.2. Sub-resources

Additionally, CityGML adopts the multi-scale modelling supporting five different LoDs (Figure 3-8). In a CityGML, the same object may be represented in different LoDs simultaneously, enabling the analysis and visualization of the same object with regard to different degrees of resolution.

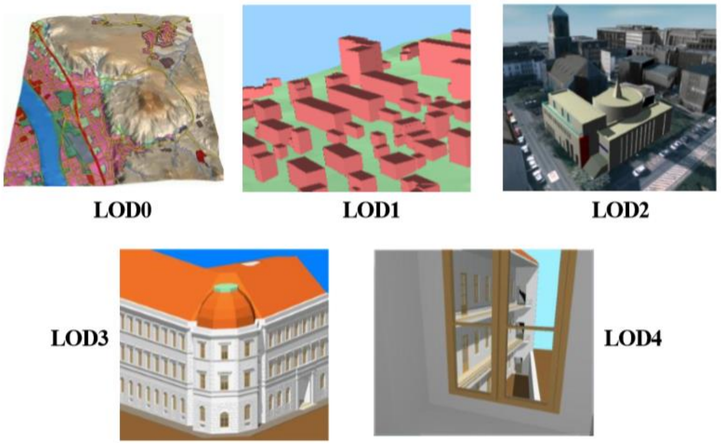


Figure 3-8: five LoD of CityGML

(Gröger et al., 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. Therefore, this semantic enrichment of each of the thematic modules is retrieved by implementing a variety of sub-resources. As a result, some of the main resources have LoD-based sub-resources and hence, their semantic retrieval is available based on the LoD (see chapter 4), while, some resources are LoD-independent and so there is no differentiation regarding their semantic sub-resources from one LoD to another (see chapter 5).

3.2.1.3. Resourse schema

The generic information retrieval schema regarding the main resources and the respective sub-resources of the CityGML RESTful Web service is schematically shown in Figure 3-9.

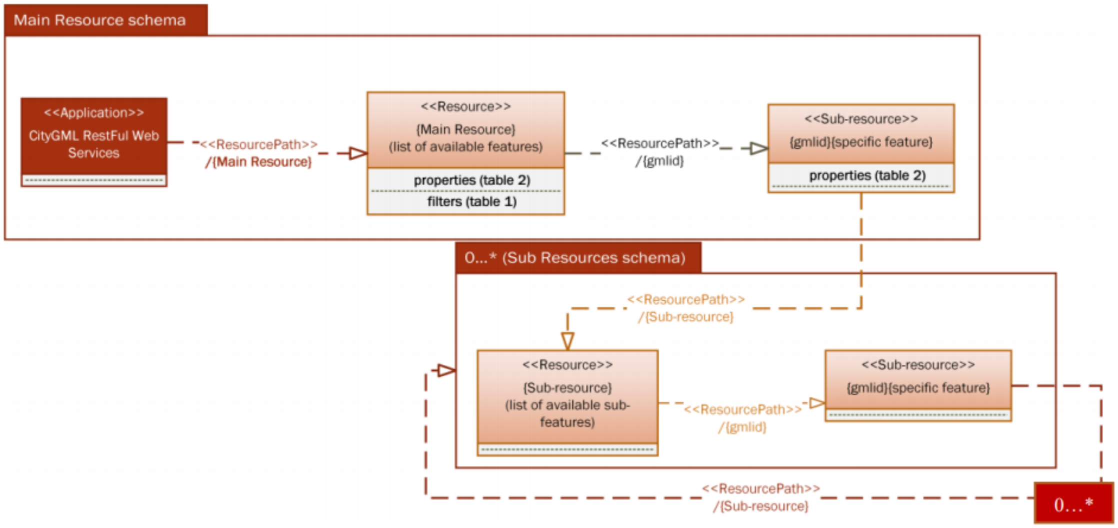


Figure 3-9: Retrieval resource schema of CityGML RESTful Web service

(Pispidikis & Dimopoulou, 2019)

This schema consists of two sub-schemas. The first one describes the retrieval mechanism of the main resources and so, the list of the main resources regarding the specific module can be retrieved using the following request:

```
../{main resources}
```

The second sub-schema describes the retrieval mechanism of the respective sub-resources. Each main schema could contain zero- to- many sub-schemas and each sub-schema could also contain zero- to- many sub-schemas.

3.2.2. ADE resources

CityGML has been designed as an application independent information model and exchange format for 3D city and landscape models. However, many applications of 3D city models require the extension by application specific feature types, attributes, and relations. For that reason, there are two available ways to extend the CityGML such as the use of generic city objects and attributes and the use of ADE.

The first concept allows for the storage and exchange of 3D objects which are not covered by any explicitly modelled thematic class within CityGML or which require attributes not represented in CityGML. These generic extensions to the CityGML are integrated into any resource of the CityGML RESTful Web service as an object attribute in the retrieved information (see section 3.2.5).

The ADE concept defines a special way of extending existing CityGML feature types which allows to use different ADEs within the same instance document simultaneously. Furthermore, each ADE is specified by its own XML schema file and is also provided with a new namespace. The integration of each ADE into CityGML RESTful Web service is not part of its core and hence, each ADE should be embedded separately as new main resource and according to its XML schema. Thereafter, the connection of the ADE resource to the desirable feature is achieved by including the ADE resource URI in the retrieved "links" object of this feature (for example see Figure 4-11; for information about the "links" object see section 3.2.5).

3.2.2.1. "Dynamizers" ADE resource implementation

Dynamizer (Chaturvedi and Kolbe, 2016) is a new concept, which extends static 3D city models by supporting variations of individual feature properties over time. Additionally, Dynamizers provide a way to model such dynamic variations with explicit time-series representations. Dynamizers also utilize standardized encodings, such as the OGC TimeseriesML standard. Utilizing this standard, the time-series can be represented as interleaved time/value pairs or by a domain range encoding with the metadata of time-series and timepoints. The time-series values may either be stored directly in-line within the CityGML document or separately in individual tables.

Chatzinikolaou, Pispidikis and Dimopoulou (2020) developed an interoperable web-based application in order to accomplish an integrated knowledge on how time-series data can be distributed in a virtual 3D environment. The methodological steps to develop the said 3D WebGIS viewer so that the available energy models can be portrayed and also the respective time series data can dynamically be retrieved based on the corresponding GML identifier (gmlid) of these models, are schematically presented in Figure 3-10.

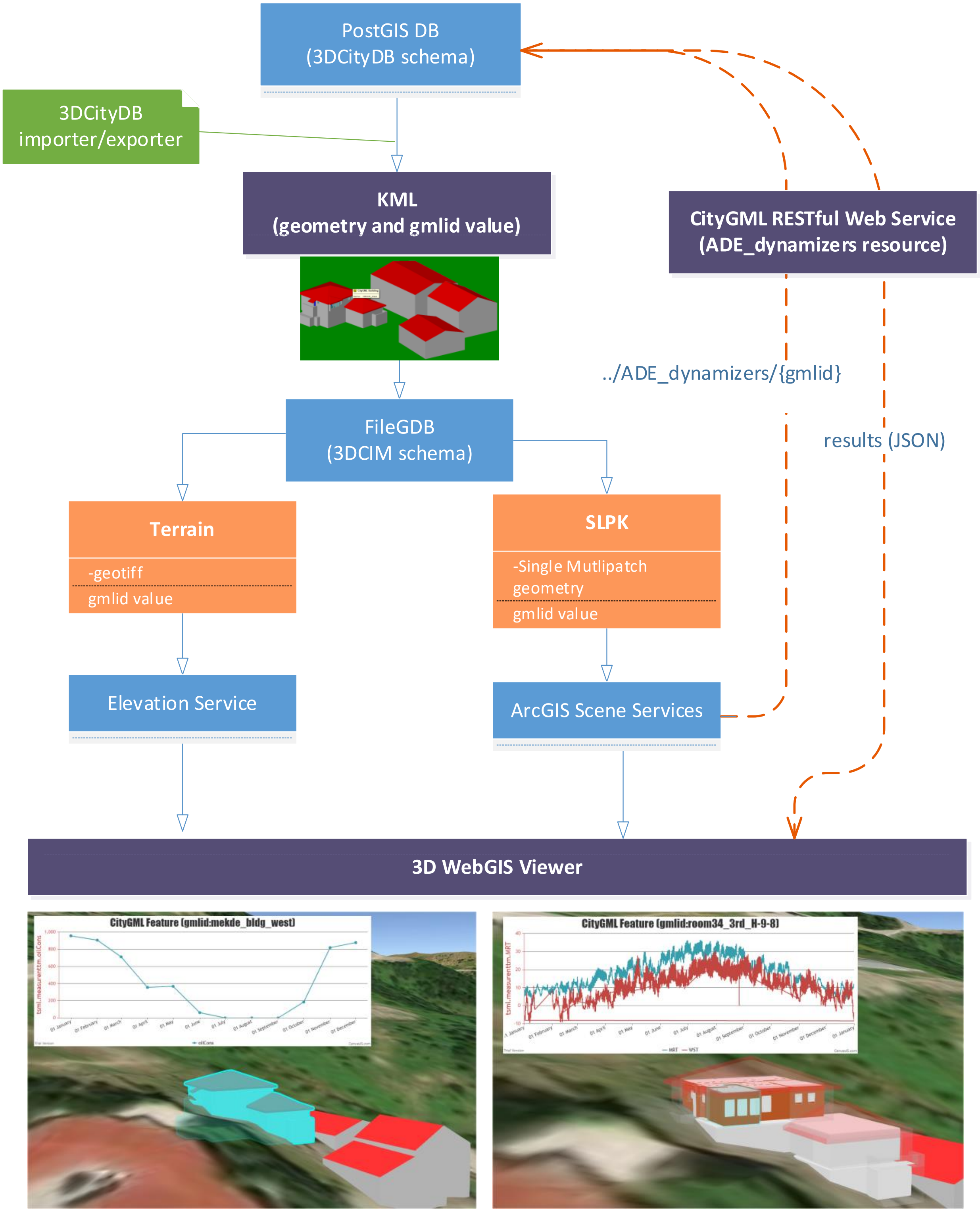
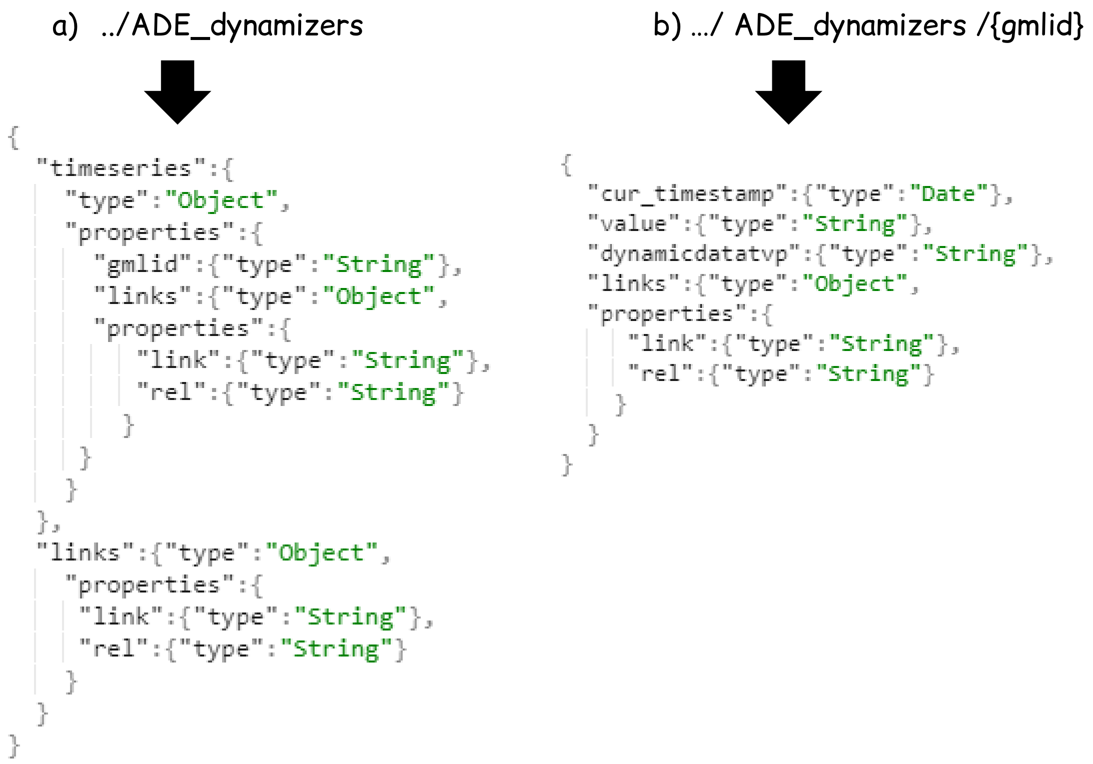

PostGIS DB
(3DCityDB schema)
3DCityDB
importer/exporter
KML
(geometry and gmlid value)
CityGML RESTful Web Service
(ADE_dynamizers resource)
../ADE_dynamizers/{gmlid}
results (JSON)
FileGDB
(3DCIM schema)
Terrain
-geotiff
gmlid value
SLPK
-Single Mutlipatch geometry
gmlid value
Elevation Service
ArcGIS Scene Services
3D WebGIS Viewer
CityGML Feature (gmlid:mekde_bldg_west)
CityGML Feature (gmlid:room34_3rd_H-9-8)

Figure 3-10: Methodological steps of the energy-based WebGIS viewer

(Chatzinikolaou et al., 2020)

The interoperable and easy-to-use time-series data retrieval is achieved by extending the CityGML RESTful Web service. Namely, the "ADE_dynamizers" main resource was embedded and thus, the available CityGML features that contain time series data can be retrieved in JSON format (Figure 3-11-(a)). Thereafter, by using the respective gmlid as sub-resource the available time-series data can be retrieved as well (Figure 3-11-(b)). The conceptual design of the "ADE_dynamizers" resource with the available properties is shown in Figure 3-12.



a) ../ADE_dynamizers

```
{
  "timeseries":{
    "type":"Object",
    "properties":{
      "gmlid":{"type":"String"},
      "links":{"type":"Object",
      "properties":{
         "link":{"type":"String"},
         "rel":{"type":"String"}
        }
      }
    }
  },
  "links":{"type":"Object",
    "properties":{
     "link":{"type":"String"},
     "rel":{"type":"String"}
    }
  }
}
```

b) .../ ADE_dynamizers /{gmlid}

```
{
  "cur_timestamp":{"type":"Date"},
  "value":{"type":"String"},
  "dynamicdatatvp":{"type":"String"},
  "links":{"type":"Object",
  "properties":{
     "link":{"type":"String"},
     "rel":{"type":"String"}
    }
  }
}
```

Figure 3-11: JSON-based schemas of "ADE_dynamizers" main resource

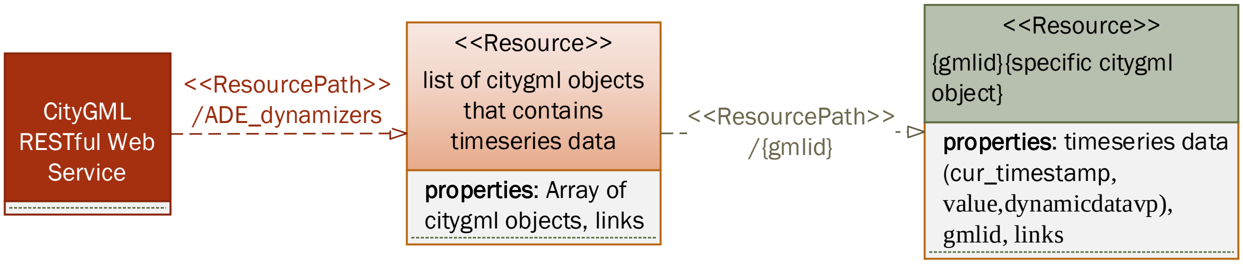

CityGML RESTful Web Service
<<ResourcePath>> /ADE_dynamizers
<<Resource>>
list of citygml objects that contains timeseries data
properties: Array of citygml objects, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid}{specific citygml object}
properties: timeseries data (cur_timestamp, value,dynamicdatavp), gmlid, links

Figure 3-12: Conceptual design of "ADE_dynamizers" main resource

3.2.3. Geometry

The Spatial properties of CityGML features are represented by GML3's geometry model, which is based on the standard ISO 19107 (ISO, 2003) and also representing 3D geometry according to the well-known Boundary Representation (B-Rep). The geometry model of GML3 consist of primitives and for each dimension, there is a geometrical primitive such as "Point" for a zero-dimensional, "Curve" for one-dimensional, "Surface" for two-dimensional and "Solid" for three-dimensional. Thereafter, a solid is bounded by surfaces and a surface by curves. Furthermore, the primitives may be combined to form complexes, composite geometries or aggregates (Figure 3-13). GML3 provides a special aggregate for each dimension such as MultiPoint, MultiCurve, MultiSurface and MultiSolid. A composite is a special complex, which can only contain elements of the same dimension such as CompositeSolid, CompositeSurface or CompositeCurve.

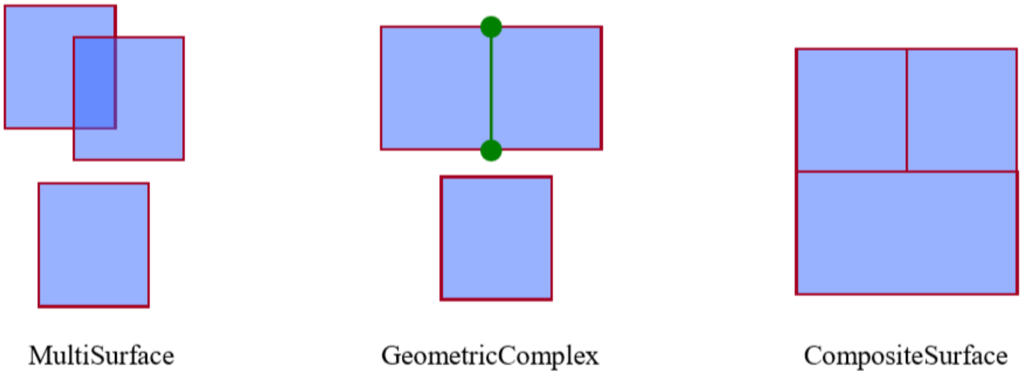


Figure 3-13: Combined geometries

(Gröger et al., 2012)

3.2.3.1. GeoJSON Implementation

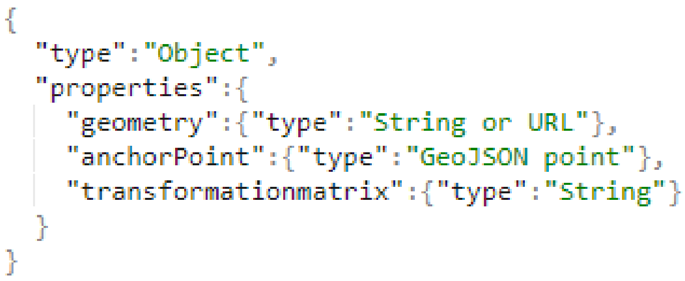
CityGML uses only a subset of the GML3 geometry package, defining a profile of GML3. Namely, in CityGML, a curve is restricted to be a straight line, thus only GML3 class "LineString" is used. Moreover, Surfaces in CityGML are represented by "Polygons", which define a planar geometry (Gröger et al., 2012). However, although the CityGML geometry is structured according to GML, the GeoJSON specification (Butler et al., 2016) will be used as a geometry retrieval format when the CityGML RESTful Web service is implemented. GeoJSON is a geospatial data interchange format based on JSON that supports a variety of geometry types. More specific, it comprises the seven concrete geometry types defined in the OpenGIS Simple Features Implementation Specification for SQL (OpenGIS, 1999) such as "Point" and "MultiPoint" for zero-dimensional, "LineString and MultiLineString for one-dimensional, Polygon and MultiPolygon for two-dimensional and GeometryCollection for heterogeneous geometries. As a result, all the available geometries of CityGML can be represented by the GeoJSON format instead of GML. Table 3-3 shows the matching supported geometries among CityGML and GeoJSON.

CityGML (GML3)	GeoJSON
Point	Point
LineString	LineString
Polygon	Polygon
MultiPoint	MultiPoint
MultiLineString	MultiLineString
MultiPolygon	MultiPolygon
Composite Polygon	MultiPolygon
Solid	MultiPolygon
MultiSolid	GeometryCollection or MultiPolygon
CompositeSolid	GeometryCollection or MultiPolygon
CompositeSurface	GeometryCollection

Table 3-3: Matching supported geometries among CityGML and GeoJSON

3.2.3.2. Implicit Object Implementation

However, in many cases, various features of a 3D city model could have same representation but different position such as tree or other vegetation objects, a traffic light or a traffic sign. The shape of these features is stored only once as a prototypical geometry which can be re-used or referenced many times, wherever the corresponding feature occurs in the 3D city model. Each occurrence is represented by an implicit object that contains a link to a prototype shape geometry (local CRS), a transformation matrix that is multiplied with each 3D coordinate of the prototype, and an anchor point denoting the base point of the object in the word Coordinate Reference System (CRS) (Figure 3-14). This principle is adopted from the concept of scene graphs used in computer graphics standards like VRLM and X3D.



```
{
  "type":"Object",
  "properties":{
    "geometry":{"type":"String or URL"},
    "anchorPoint":{"type":"GeoJSON point"},
    "transformationmatrix":{"type":"String"}
  }
}
```

Figure 3-14: JSON-based schema of the implicit object

In conclusion, all the available geometries of CityGML can be represented either by GeoJSON format or by implicit object.

3.2.4. General filters

The response of each request by implementing the main resources of the CityGML RESTful Web service is mainly a list of the available thematic modules respectively. Each thematic modules of this list contains general information according to CityGML specification.

Moreover, most of the main resources can be filtered using the general filters such as function, usage, class, bbox and lod (Table 3-4).

Filter	URI (Example by using brid resource)
function	../ brid?function=3020
usage	../ brid?usage=1010
class	../ brid?class =1000
bbox	../ brid?bbox= 334433, 4455667, 445677, 5566556
lod	../ brid?lod=3

Table 3-4: General filters of main thematic resources

The attributes class, function and usage are available for almost all CityGML feature types and their values are specified in code lists, which are implemented as simple dictionaries following the GML3.1.1 simple Dictionary Profile (Whiteside, 2005). Additionally, their content may substantially vary for different countries (e.g. due to national law or regulations) and for different information communities and therefore, the international standard GML does not specify normative code lists for any of the attributes of type "gml:CodeType"[14]. However, a non-normative code lists for selected attributes were proposed and maintained by the Special Interest Group 3D (SIG 3D) of the GDI-DE. These code lists can be directly referenced in CityGML instance documents and serve as an example for the definition of code lists (Gröger et al., 2012). For instance, according to this code list, the code value 2000 for the building attribute function is referred as a post office. Moreover, the bbox filter parameters is vital to be defined so that the retrieval data to be filtered based on spatial queries. Furthermore, since the semantic richness of CityGML is based on the available LoD, the LoD should be defined as general filter.

[14] **gml:CodeType**: In case a fixed enumeration of possible attributes values is not suitable, the attribute type is specified as gml:CodeType and the allowed attribute values can be provided in a code list which is specified outside the CityGML schema.

3.2.5. Information retrieval

The information retrieval about a specific main resource can be achieved implementing the respective gmlid as sub-resource.

```
../{main resources}/{gmlid}
```

CityGML is explicitly designed as topographic-based model (Gröger & Plümer, 2012). However, many applications of 3D city models require the extension by application specific feature types, attributes, and relations. The generic city objects and attributes are an alternative mechanism to extend CityGML providing ad hoc solutions for the storage and exchange of 3D objects which are not covered by any explicitly modeled thematic class within CityGML or required attributes not represented in CityGML. Since this approach is ad hoc, no application schema is required. Consequently, the generic object as a retrieval property provides an ad hoc list of key value pairs based on the generic model of CityGML. Moreover, the "links" object is vital to be provided as retrieval property so that the HATEOAS implementation is achieved and then the CityGML Web service to be RESTful. As a result, each resource should contain information regarding links to other available resources. Consequently, the "links" object of a resource of CityGML RESTful Web service contains a list of key value pairs links to itself, to all parents' resources and to a child resource. In addition, the "geometry" object can be retrieved based on GeoJSON specification (see section 3.2.2), external format such as X3D, COLLADA etc, or implicit object. Moreover, the said object is only available when no additional sub-resources of a particular feature exists. For Example, in LoD0-1, the bldg main resource contains the geometry object, while, in LoD2-4 the geometry object is only available in the last existing sub-resource. Another retrieval property is the "address" object, which contains information that is specified using the Extensible Address Language (xAL) address standard by the OASIS consortium (OASIS, 2003) providing a generic schema for all kinds of international addresses. Also, the "*XXX*Information" object can be retrieved including a variety of information based on the respective CityGML module. The characters "XXX" describe the respective name of the main resources (e.g. bridInformation, bldgInformation). Additionally, the "lod" attribute can be retrieved providing information about the level of detail of the retrieval data. Last but foremost, two Boolean attributes are retrieved such as "isMovable" and "XXXPart", which inform the client about whether the retrieval object is movable or not and whether it is XXXPart or not e.g. tunnel part, building part or bridge part.

In conclusion, the general but not mandatory value types of the retrieval properties with respect to main resources of the CityGML RESTful Web service are described in Table 3-5

Information	Type	Description
lod	Number	LoD value
XXXPart*	Boolean	True or False
isMovable	Boolean	True or False
XXXInfomation*	Object	List of key value pairs based on respective module
geometry	Object	Geometry object based on GeoJSON specification, external format or implicit object
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

**The characters "XXX" are defined based on the name of the main resource (e.g bridsPart, bridInformation)*

Table 3-5: Available information of the main resources

3.2.6. Security

Unlike WS-* that specifies a well-defined security model that is protocol independent and is built specifically for SOAP Web services, REST does not currently have its own security model. Instead, today's REST security best practices leverage existing HTTP security implementation approaches (Sudhakar, 2011). Fortunately, there are various HTTP approaches for securing web applications such as HTTP Basic Authentication, HTTP Digest Authentication and Token Based Authentication (OAuth). The HTTP Basic Authentication mainly uses the ID and password of a client to authenticate the client's request in HTTP header. Since, the Client's ID and password get encoded with Base64, which is stored in the HTTP Authenticated header without being encrypted or hashed, they are usually sent over HTTPS or Secure Sockets Layer (SSL). However, this approach has security vulnerabilities of replay attack, injection attack and middleware hijacking (Jo, Kim, & Lee, 2014). The advanced version of the first approach is that of HTTP Digest Authentication, which encrypts the clients' ID and password via hash such as MD5 (Peng, Li, & Huo, 2009). It should be noted that this approach can be exposed to a Man-in-the-Middle attack, also known as a hijack attack. Finally, the Token Based Authentication (OAuth) uses a token instead of user's ID and password (Jo, Kim, & Lee, 2014). Consequently, the use of a token in communication between a user and Resource Server does not expose the user's ID and password and thus, this approach is frequently implemented by various Web service companies such as Twitter, Yahoo, Google, Facebook, Microsoft etc.

However, it should be noted that since the REST is based on the principle of statelessness, the aforementioned security approaches have to authenticate every single request of a client each time.

3.2.7. Cross-Domain issues

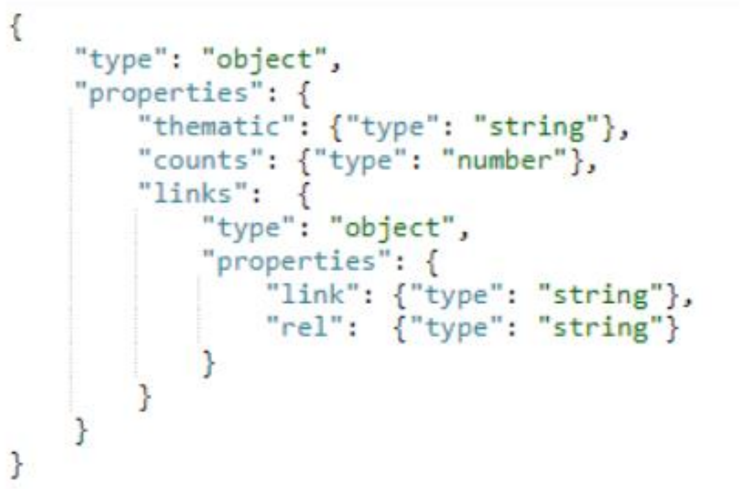
The execution of each request of CityGML RESTful Web Service is implemented in accordance with HTTP specification. Hence, for the retrieval of the data the HTTP GET method is implemented (Pispidikis and Dimopoulou, 2018). The utilization of this mechanism from distributed resources with different domains may have Cross-Domain issues. These issues mean that certain Cross-Domain requests will be forbidden by default by the same-origin security policy (W3C, 2010). Fortunately, the modern browsers support several techniques for overcoming these issues such as CORS (Cross-Origin Resource Sharing) and JSONP (JSON with Padding). The CORS is considered a standard and a mechanism that allows JavaScript on a web page to consume REST API served from a different origin. The CORS can be implemented through the HTTP Header "Access-Control-Allow-Origin", which can be enabled in a RESTful Web Service. An alternative way to share the data bypassing the same-origin policy without need of modern browser is the JSONP format which does not use the XMLHttpRequest object. Instead, it dynamically inserts <script> tag into a webpage that is not considered as Cross-Domain issue. However, apart from the aforementioned solutions, a proxy server can be utilized when executing the desirable request, avoiding all issues regarding the Same-Origin Policy. More specific, a proxy server can receive any request from distributed resource and then acting as a client on behalf of the user, requests the data from the server. Thereafter, when the data is returned, the proxy server relates it to the origin request and forwards it to the user.

3.3. Citymodels and Gmlid Resources

3.3.1. Citymodels resource

A SOAP-based Web service provides an XML-based interface description language called WSDL that is used to describe the functionalities offered by this Web service. Hence, users are able to have an overview of all these functionalities. On the other hand, REST does not provide any standards like WSDL to inform users of its available endpoints. Consequently, CityGML RESTful Web service should enable users to have an overview of the available thematic models by defining a core resource. This resource is called "citymodels" and is mainly used to retrieve the total number of the available thematic models grouped by thematic category model. In each group category, the corresponding resource link of the main thematic resource (see Table 3-2) is also be retrieved and thus, users can send additional requests and receive more specific data.

The JSON-based schema of the retrieval data by implementing the "citymodels" resource is shown in Figure 3-15.



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

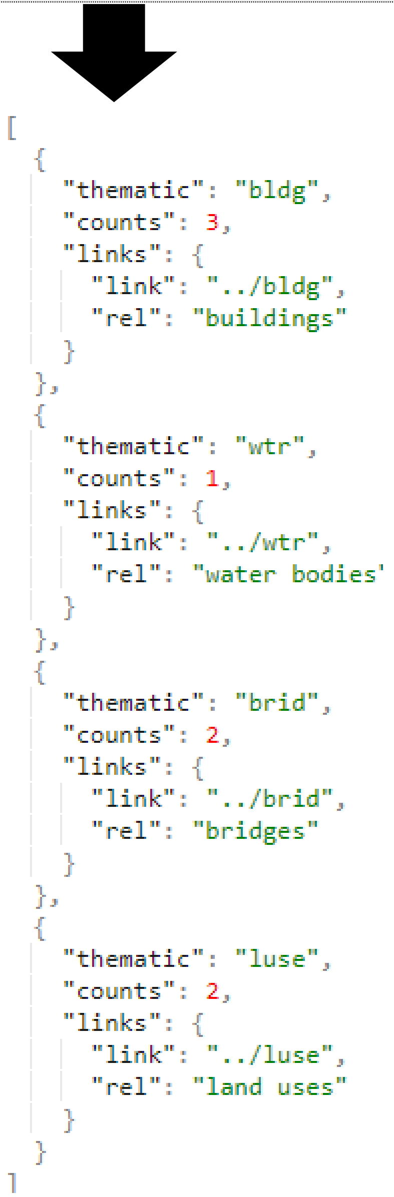
Figure 3-15: citymodels resource schema in JSON format

(Pispidikis & Dimopoulou, 2018)

3.3.1.1. Case study using the “citymodels” resource

A CityGML dataset contains a variety of thematic modules in different LoDs such as three buildings, one waterbody, one bridge (LoD2 & LoD3) and two land uses. So, the implementation of the “citymodels” resource retrieves the following response (Figure 3-16)

../citymodels



```
[
  {
    "thematic": "bldg",
    "counts": 3,
    "links": {
      "link": "../bldg",
      "rel": "buildings"
    }
  },
  {
    "thematic": "wtr",
    "counts": 1,
    "links": {
      "link": "../wtr",
      "rel": "water bodies'
    }
  },
  {
    "thematic": "brid",
    "counts": 2,
    "links": {
      "link": "../brid",
      "rel": "bridges"
    }
  },
  {
    "thematic": "luse",
    "counts": 2,
    "links": {
      "link": "../luse",
      "rel": "land uses"
    }
  }
]
```

Figure 3-16: JSON result by using "citymodel" resource

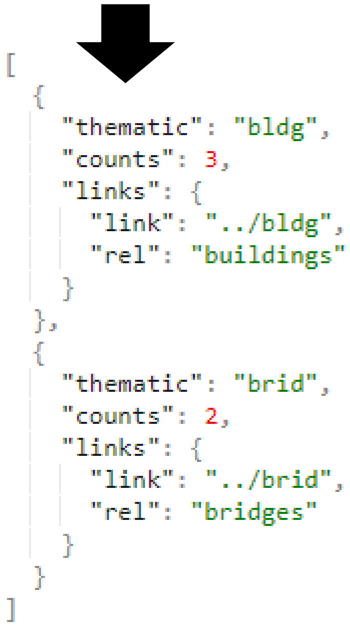
It should be noted that in the above mentioned example, although bridge module has one bridge, there are two instances of this bridge based on the corresponding LoD.

3.3.1.2. Filters

The "citymodels" resource retrieves information regarding the available CityGML thematic modules and 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 value of this filter is based on the respective namespace prefix of the thematic modules of CityGML v2 specification (see Table 3-2). Also, multi thematic values can be used simultaneously by separating them using comma punctuation.

By using the same dataset of the previous case study (see section 3.3.1.1), then if a user only needs information about the available buildings and bridges the following request can be implemented (Figure 3-17).

```
../citymodels?thematics=bldg,brid
```



```
[
  {
    "thematic": "bldg",
    "counts": 3,
    "links": {
      "link": "../bldg",
      "rel": "buildings"
    }
  },
  {
    "thematic": "brid",
    "counts": 2,
    "links": {
      "link": "../brid",
      "rel": "bridges"
    }
  }
]
```

Figure 3-17: JSON result by using the "thematics" filter parameter in the "citymodels" resource

In Addition, the filter "bbox" is defined. The value of this filter is a geometry rectangle in a specific reference system which limits the results according to a boundary box. Another important filter parameter is the CRS of the data called "epsg". The definition of the spatial reference system is of utmost importance and a key requirement for the integration of different spatial datasets in a single 3D city model. CityGML inherits GML3's spatial capabilities of handling CRS. More specific, in CityGML, the coordinate system of the geometries is defined through the attribute "srsName" which is inherited from the abstract GML superclass "gml:_Geometry". The value of this attribute may be a reference to a Well-known CRS definition provided by an authority organization such as the European Petroleum Survey Group (EPSG) (Pispidikis & Dimopoulou, 2015), but may also be a pointer to a CRS that is locally defined within the same CityGML instance document. The value of this pointer is based on the Uniform Resource Name (URN) encoding standard (Whiteside, 2009) having the following generic syntax:

```
urn:ogc:def:objectType,objectType:authority:version:code,objectType:authority:versio
n:code
```

In case that there is a CityGML dataset in which two reference system should be defined e.g. EPSG:25832 for projected CRS and EPSG:5783 for vertical CRS then the following URN is formed:

```
urn:ogc:def:crs,crs:EPSG::25832,crs:EPSG::5783
```

The replacements between the general URN syntax and the CRS is presented in Table 3-6. Consequently, when the "epsg" filter is not used, then the default CRS is set based on CityGML dataset. On the other hand, when "bbox" filter uses different CRS then this CRS should be set as a value to the "epsg" filter.

URN general syntax	CRS
objectType	crs
Authority	EPSG
Version	-
Code	25832 & 5783

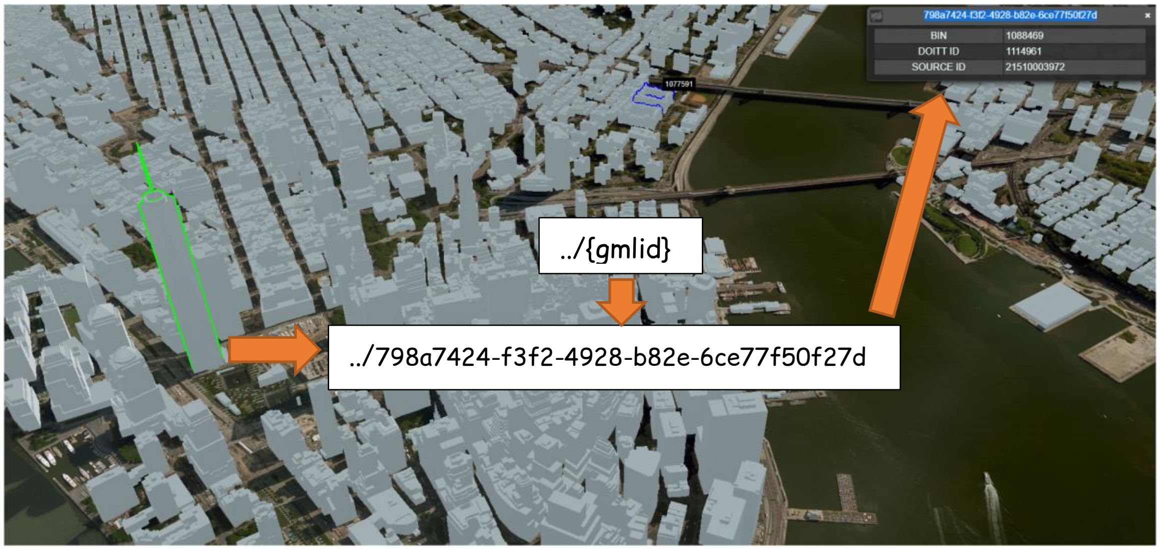
Table 3-6: URN syntax for CRS references

The available filters of "citymodels" resource can be implemented simultaneously and thus, the spatial and descriptive filtering of the desired request can be achieved. The following request only provides information on the amount of buildings and bridges that are located within the specified boundary box in EPSG:3857 CRS.

```
../citymodels?thematics=bldg,brid&bbox=334433, 4455667, 445677, 5566556&epsg=3857
```

3.3.2. Gmlid resource

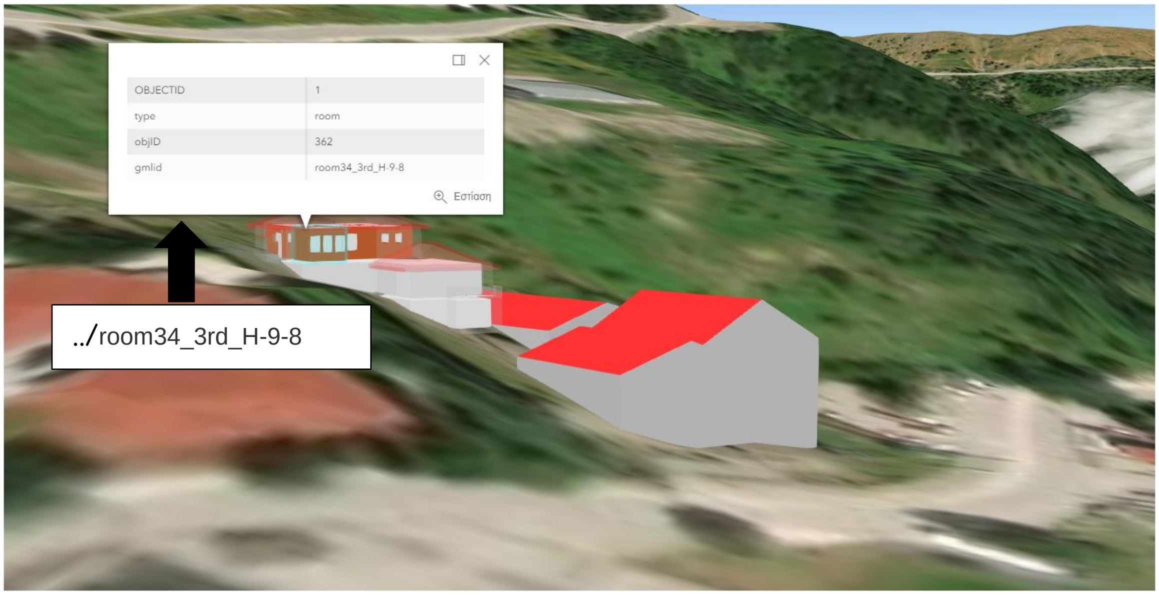
In some cases, the retrieval of the available information about a semantic feature is most useful to be achieved directly based on the respective gmlid. This capability could be possibly utilized at the scenario of 3D Web visualization of huge CityGML models, where the user needs to retrieve the available descriptive information based on the respective gmlid value of the selected model. More specific, after the CityGML models are converted to the appropriate format for 3D web visualization (see section 2.1), they should have only the respective gmlid value as attribute. Thereafter, the gmlid sub-resource can be implemented and the rest of the available information to be retrieved (Figure 3-18).



798a7424-f3f2-4928-b82e-6ce77f50f27d

BIN	1088469
DOITT ID	1114961
SOURCE ID	21510003972

(a) Information retrieval of LoD1 building



OBJECTID	1
type	room
objID	362
gmlid	room34_3rd_H-9-8

(b) Information retrieval of a building room in LoD4

Figure 3-18: Information retrieval based on the gmlid by using "gmlid" resource

4. LOD-BASED THEMATIC RESOURCES

4.1 Bldg Thematic Resource

4.2 Tun Thematic Resource

4.3 Brid Thematic Resource

This Chapter describes the conceptual design of the LoD-based thematic resources of the CityGML RESTful Web service. More specific, the "bldg", "tun" and "brid" resources and their respective child resources are presented. Also, for each of these resources, various case studies using semantic requests are presented.

In this chapter, the 4th sub-research question of the current dissertation is partially answered:

> *How could CityGML data be semantically retrieved by users without knowledge of the source?*

This chapter is based on the following papers:

(1) **Pispidikis** and Dimopoulou (2018)
(2) Athanasiou, **Pispidikis** and Dimopoulou (2018)
(3) **Pispidikis** and Dimopoulou (2019)

4.1. Bldg Thematic 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 (Gröger & Plümer, 2012). Spanning the different levels of detail, the building model differs in the complexity and granularity of the geometric representation and the thematic structuring of the model into components with a special semantic meaning (Figure 4-1).

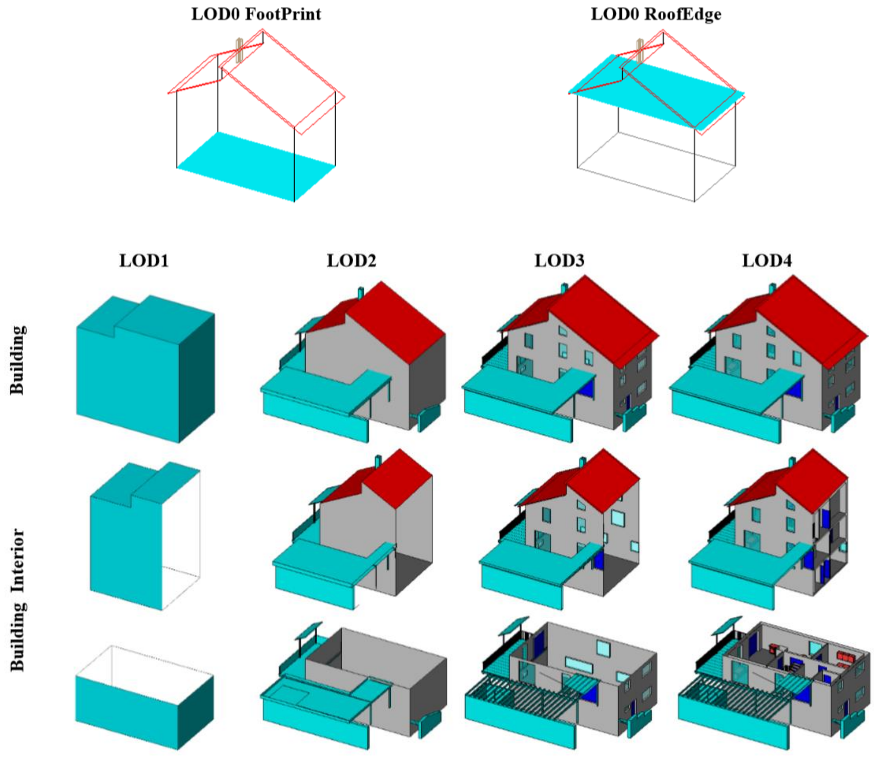


Figure 4-1: Building module in different LoD

(Gröger et al., 2012)

More specifically, in LoD0 the building is represented by horizontal surfaces describing the footprint and the roof edge. In LoD1, the different structural entities of a building are aggregated to a simple block and not differential in detail. In LoD2 and higher LoD, the exterior shell of a building can also be composed of semantic objects. Table 4-1 provides an overview of the semantic availability of a building per LoD.

	Geometric/ semantic theme	LoD0	LoD1	LoD2	LoD3	LoD4
	Footprint and roof edge	■				
	Volume part of the building shell		■	■	■	■
	Building parts		■	■	■	■
Semantic themes	Boundary surfaces			■	■	■
	Outer building installations			■	■	■
	Openings				■	■
	Rooms					■
	Interior building installation					■

Table 4-1: Semantic availability of a building per LoD

Taking into consideration the aforementioned semantic availability, the building module is enriched by semantic characteristics from LoD2 and above and thus, the child resources of the bldg main resource are defined based on the semantic enrichment of building features from LoD2 to LoD4.

4.1.1. Bldg main resource

The main thematic resource regarding building module of CityGML is the bldg resource. This resource retrieves the available building and building parts respectively, including their available information (see Table 3-5) as well as a link object that contains the URIs of these child resources and the URI of the bldg main resource. According to CityGML v2 specification, the pivotal class of the building model is "_AbstractBuilding" which is specialized either to a "Building" or to a "BuildingPart". Both these classes inherit the attributes of "_AbstractBuilding" such as the class of the building, the function (e.g. residential, public, or industry), the usage, the year of construction, the year of demolition, the roof type, the measure height and the number and individual heights of the storeys above and below the ground (Gröger et al., 2012). The available values of the latter are described in Figure 4-2.

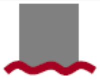
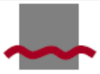
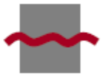
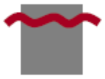
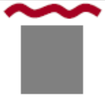
relativeToTerrain value	Illustration
entirelyAboveTerrain	
substantiallyAboveTerrain	
substantiallyAboveAndBelowTerrain	
substantiallyBelowTerrain	
entirelyBelowTerrain	

Figure 4-2: Values of the relationship of an object to the terrain

(Gröger & Plümer, 2012)

All these attributes belong to the bldgInformation object. Additionally, if a building consists of one homogeneous part, the value of the bldgPart attribute is false. On the other hand, when a building composes of different structural segments, for example a number of storeys or roof type, then the respective bldgPart value is true since the building has to be separated into a building that has one or more additional building parts (Figure 4-3).

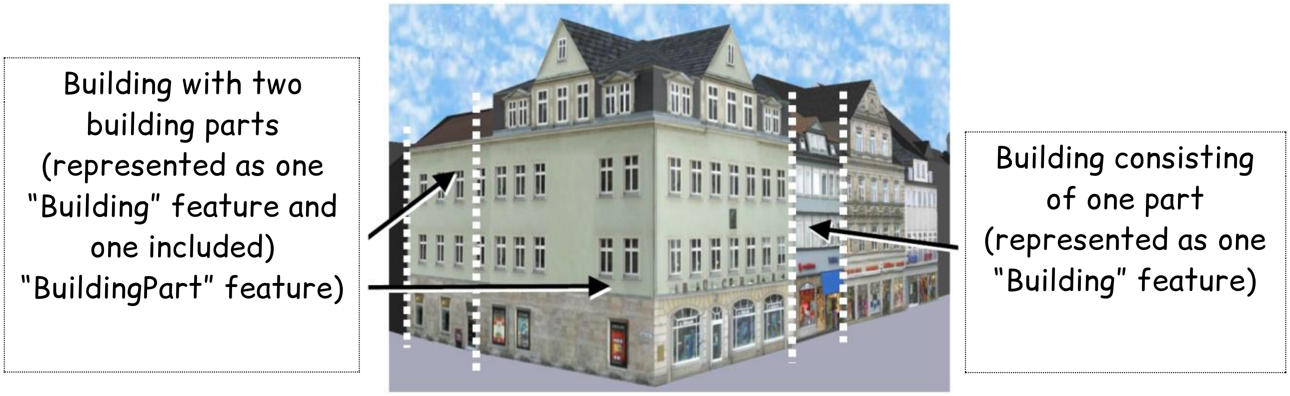

Building with two building parts (represented as one "Building" feature and one included) "BuildingPart" feature)
Building consisting of one part (represented as one "Building" feature)

Figure 4-3: Examples of building consisting of one and two building parts

(Gröger et al., 2012)

Another capability of the bldg resource is the implementation of several filters that limit the retrieval result avoiding the over and under fetching issues. Except for the general filters (see Table 3-4), which are used in all main thematic resources, the bldgPart filter should be defined. The value of this filter is Boolean and limits the retrieval result on whether the building is building part or not.

The conceptual UML model of the bldg main resource with available properties and filters is shown in Figure 4-4

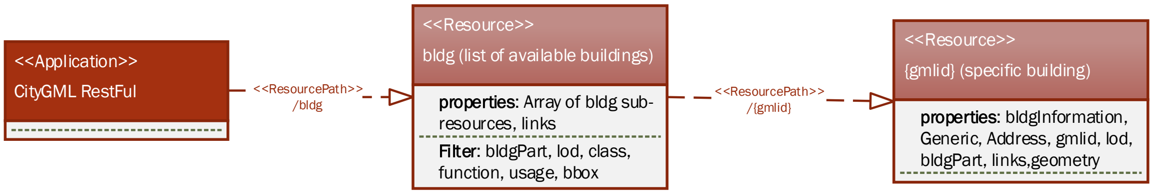
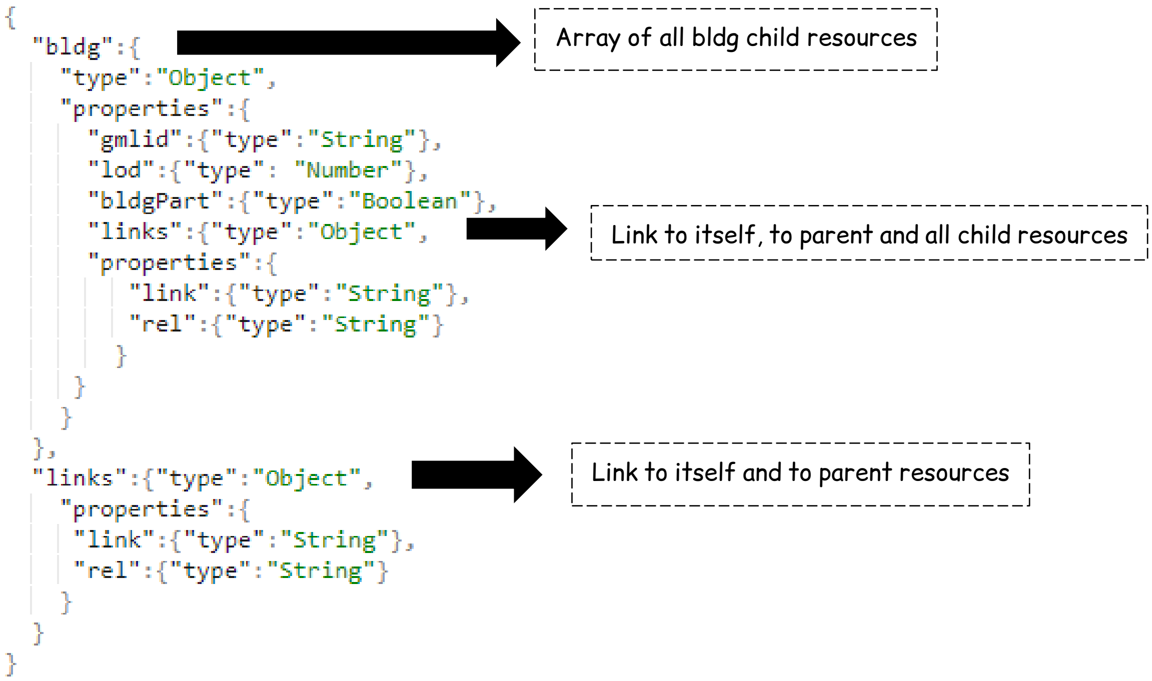

<<Application>>
CityGML RestFul
<<ResourcePath>>
/bldg
<<Resource>>
bldg (list of available buildings)
properties: Array of bldg sub-resources, links
Filter: bldgPart, lod, class, function, usage, bbox
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific building)
properties: bldgInformation, Generic, Address, gmlid, lod, bldgPart, links,geometry

Figure 4-4: Conceptual design of the "bldg" main resource

The schema of the bldg main resource in JSON format is shown in Figure 4-5



```
{
  "bldg":{
    "type":"Object",
    "properties":{
      "gmlid":{"type":"String"},
      "lod":{"type": "Number"},
      "bldgPart":{"type":"Boolean"},
      "links":{"type":"Object",
      "properties":{
         "link":{"type":"String"},
         "rel":{"type":"String"}
        }
      }
    }
  },
  "links":{"type":"Object",
    "properties":{
     "link":{"type":"String"},
     "rel":{"type":"String"}
    }
  }
}
```

Array of all bldg child resources

Link to itself, to parent and all child resources

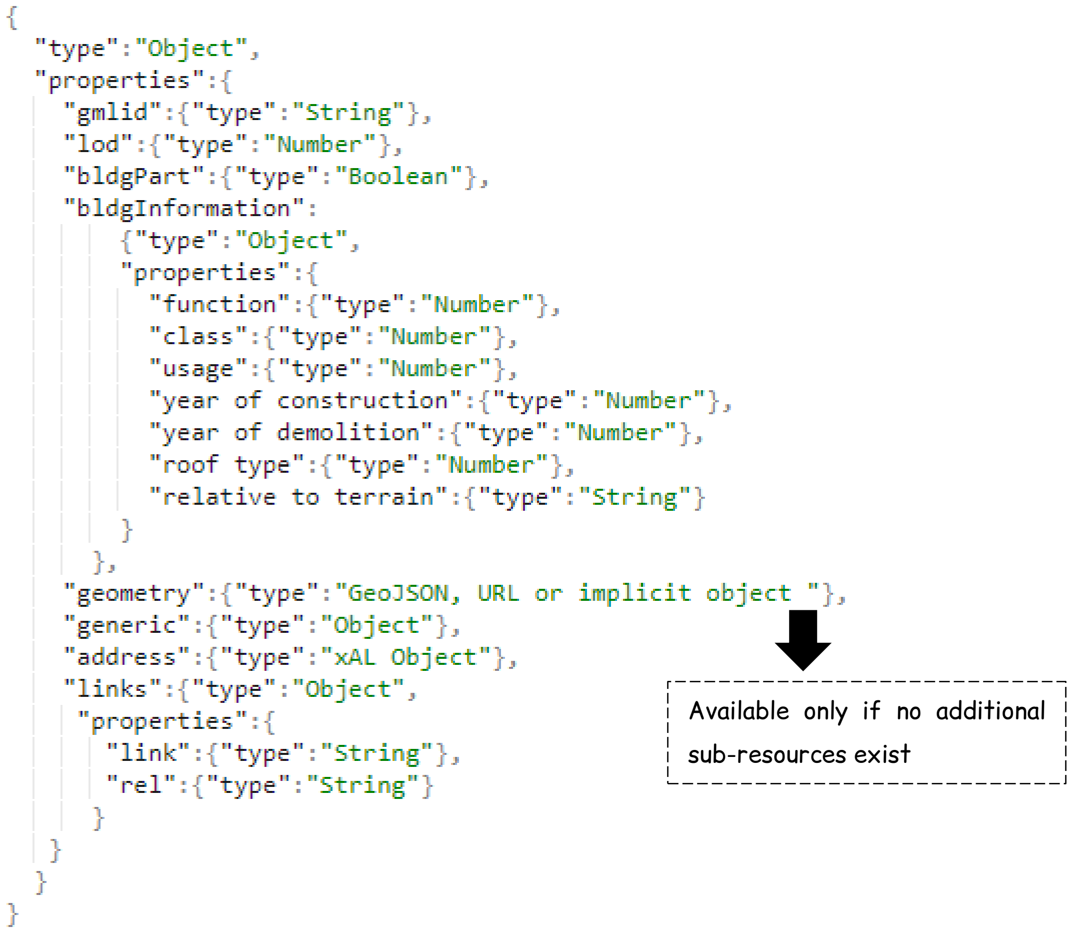
Link to itself and to parent resources

Figure 4-5: bldg resource schema in JSON format

Also, the retrieval of a specific building can be achieved by implementing the gmlid as sub-resource.

```
../bldg/{gmlid}
```

The gmlid in the brackets is the unique id for each building according to CityGML. This sub-resource of a particular building contains additional information as opposed to the bldg main resource (see Figure 4-5) so that the overfetching of the data is avoided. As a result, the JSON-based retrieval information is presented in Figure 4-6



```
{
  "type":"Object",
  "properties":{
    "gmlid":{"type":"String"},
    "lod":{"type":"Number"},
    "bldgPart":{"type":"Boolean"},
    "bldgInformation":
        {"type":"Object",
        "properties":{
          "function":{"type":"Number"},
          "class":{"type":"Number"},
          "usage":{"type":"Number"},
          "year of construction":{"type":"Number"},
          "year of demolition":{"type":"Number"},
          "roof type":{"type":"Number"},
          "relative to terrain":{"type":"String"}
        }
      },
    "geometry":{"type":"GeoJSON, URL or implicit object "},
    "generic":{"type":"Object"},
    "address":{"type":"xAL Object"},
    "links":{"type":"Object",
     "properties":{
       "link":{"type":"String"},
       "rel":{"type":"String"}
      }
    }
  }
}
```

Available only if no additional sub-resources exist

Figure 4-6: JSON-based resource schema of specific building

4.1.2. LoD2 bldg sub-resources

The supported semantic characteristics of the LoD2 building are the exterior boundary surfaces such as Wallsurface, RoofSurface, GroundSurface, OuterCeilingSurface and OuterFloorSurface as well as the exterior building installation (see Table 4-1). The exterior boundary surfaces are implemented to semantically structure the exterior shell of building (Figure 4-7). Specifically, the ground plate of a building is modeled by the GroundSurface. In addition, the mostly horizontal surface that belongs to the outer shell and also has the orientation pointing downward such as the visible part of the ceiling of a loggia or the ceiling of a passage, modeled by the OuterCeilingSurface. Furthermore, the OuterFloorSurface is utilized to model the mostly horizontal surface that belongs to the outer building shell and with the orientation point upwards such as the floor of a loggia. Moreover, all parts of a building façade belonging to the outer building shell can be modeled by the WallSurface. Also, the RoofSurface is used to express the major roof parts of a building whilst secondary parts of a roof with specific semantic meaning like dormers or chimneys are modeled as exterior building installation. The exterior building installation is an outer component of a building which has not the significance of a BuildingPart, but it strongly affects the outer characteristic of the building.

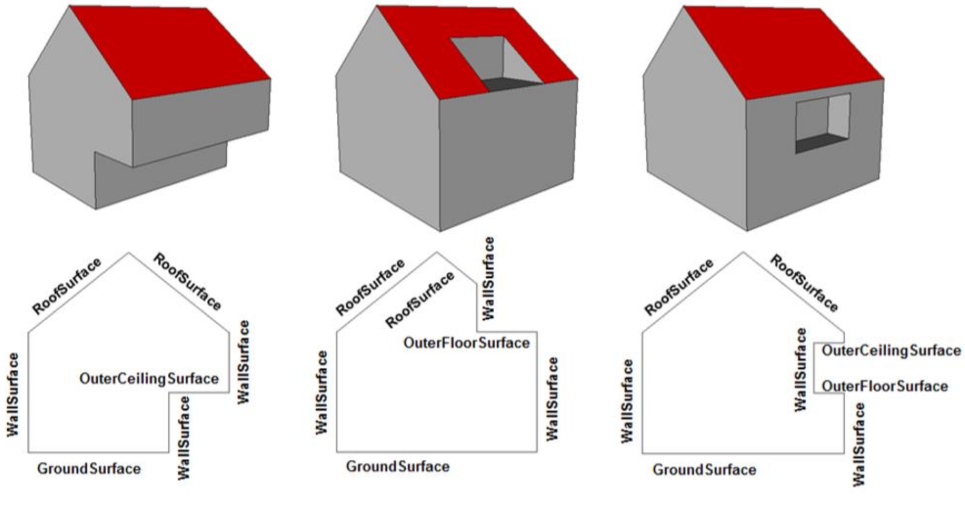


Figure 4-7: Boundary surfaces of the outer building shell

(Gröger et al., 2012)

Consequently, the aforementioned semantic features are the LoD2 child resources of the bldg main resource. The URI resources regarding the exterior boundary surfaces are walls, roofs, grounds, ceilings and floors respectively. Additionally, the exterior building installation resource is called "installation". This resource can be filtered using a variety of filters such as usage, function, class and type. It should be noted that the "installation" resource refers to both interior and exterior building installation. 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 installation are semantic features available in LoD4. So, the "installation" resource is defined as a child sub-resource regarding LoD4 as well. Furthermore, the "closure" resource is embedded so that the open sides of the building can be virtually closed by using the ClosureSurface. 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 4-2

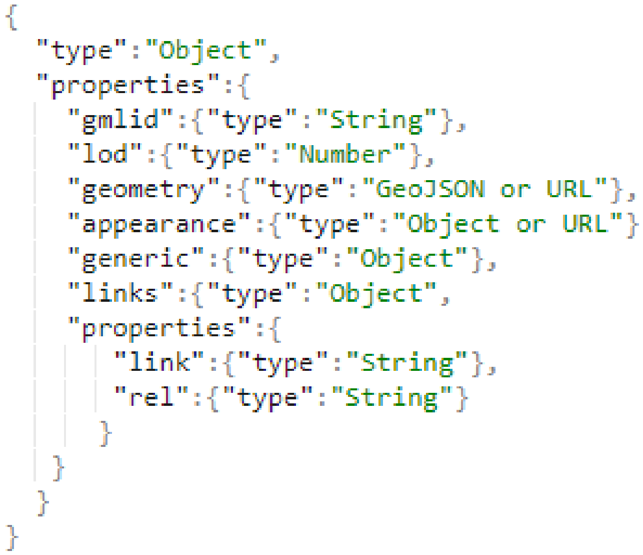
Information	Type	Resource	Description
lod	Number	Installations, Exterior boundaries*	LoD value
appearance	Object	Installations, Exterior boundaries*	List of key value pairs based on appearance module
geometry	Object	Installations, Exterior boundaries*	Geometry object based on GeoJSON specification or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

links	Object	Installation, Exterior boundaries*	List of key value pairs regarding links to itself, to parent and to child resources
gmlid	String	Installation, Exterior boundaries*	gmlid value
usage	Number	installation	Codelist
function	Number	installation	Codelist
class	Number	installation	Codelist
type	Number	installation	exterior or interior

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

Table 4-2: Available information of LoD2 bldg sub-resources

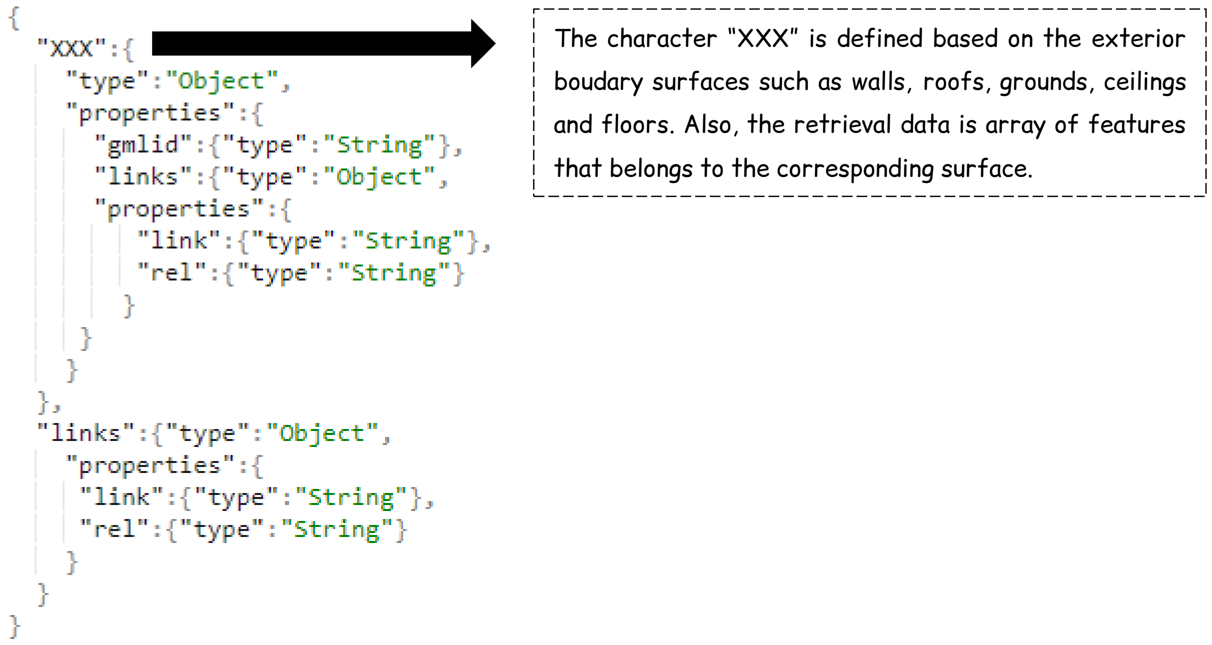
The exterior boundary surface sub-resources with regard to a specific feature (or gmlid) have a similar schema, which is presented in Figure 4-8 in JSON format.



```
{
  "type":"Object",
  "properties":{
    "gmlid":{"type":"String"},
    "lod":{"type":"Number"},
    "geometry":{"type":"GeoJSON or URL"},
    "appearance":{"type":"Object or URL"}
    "generic":{"type":"Object"},
    "links":{"type":"Object",
    "properties":{
       "link":{"type":"String"},
       "rel":{"type":"String"}
      }
    }
  }
}
```

Figure 4-8: Schema of the Bldg exterior boundary surface sub-resources in JSON format

Also, the retrieval schema of features belonging to a specific bldg exterior boundary surface category e.g. walls, is presented in Figure 4-9 (for the JSON-based schema for each boundary surface see Annex A.1)



```
{
  "XXX":{
    "type":"Object",
    "properties":{
      "gmlid":{"type":"String"},
      "links":{"type":"Object",
      "properties":{
         "link":{"type":"String"},
         "rel":{"type":"String"}
        }
     }
    }
  },
  "links":{"type":"Object",
    "properties":{
     "link":{"type":"String"},
     "rel":{"type":"String"}
    }
  }
}
```

The character "XXX" is defined based on the exterior boudary surfaces such as walls, roofs, grounds, ceilings and floors. Also, the retrieval data is array of features that belongs to the corresponding surface.

Figure 4-9: Retrieval schema of features belonging to a specific bldg exterior boundary surface

The retrieval information of the "appearance" object is based on the appearance module of CityGML. This module is not limited to visual data but represents arbitrary categories called "themes" such as infrared radiation, noise pollution, or earthquake-induced structural stress (Gröger et al., 2012). Consequently, a single surface geometry may have surface data for multiple themes. As a result, the "appearance" object could contain an array of themes whose value depends on the appearance module of CityGML e.g. color or URL to an image. Additionally, the "links" object of a sub-resource is used to provide information about the URIs of itself, to parent and to all child resources. For example, there is a building (gmlid:1) in LoD2 which has four WallSurfaces, one GroundSurface and two RoofSurfaces (Figure 4-10).

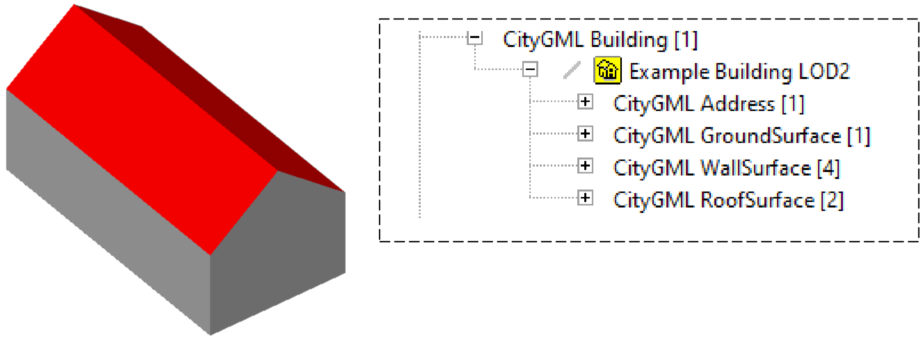
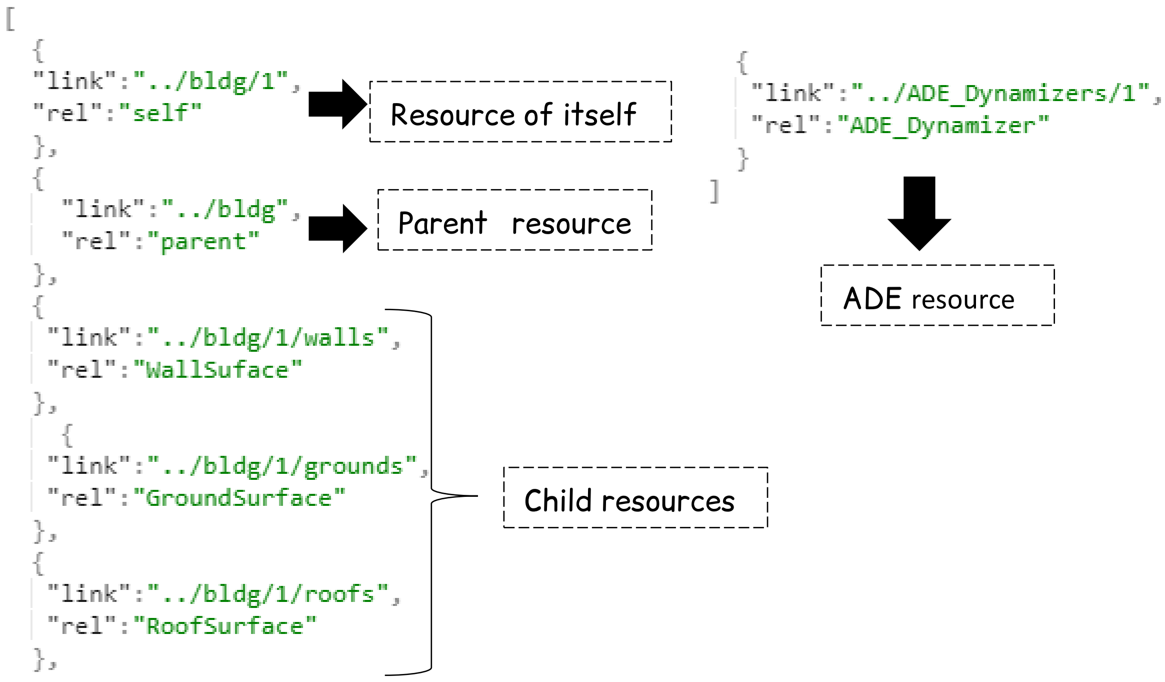


Figure 4-10: Example of a LoD2 building

So, if a client makes a request using the sub-resource below:

```
../bldg/1
```

then, with regard to the "link" object the following information is retrieved in JSON format (Figure 4-11):



```
[
  {
  "link":"../bldg/1",
  "rel":"self"
  },
  {
    "link":"../bldg",
    "rel":"parent"
  },
  {
   "link":"../bldg/1/walls",
   "rel":"WallSuface"
  },
    {
   "link":"../bldg/1/grounds",
   "rel":"GroundSurface"
  },
  {
   "link":"../bldg/1/roofs",
   "rel":"RoofSurface"
  },
  {
   "link":"../ADE_Dynamizers/1",
   "rel":"ADE_Dynamizer"
  }
]
```

Figure 4-11: Retrieval data regarding the "link" object of a building in JSON format

As a result, the user could be informed not only about the existence of the exterior boundary surfaces of a building but also about the respective URIs.

The "installation" resource schema has a similar structure to exterior boundary surfaces apart from the fact that four additional attributes are included such as type, usage, function and class (see Annex A.2).

In conclusion, the conceptual design of the bldg resource with available properties and filters according to LoD2 is shown in Figure 4-12.

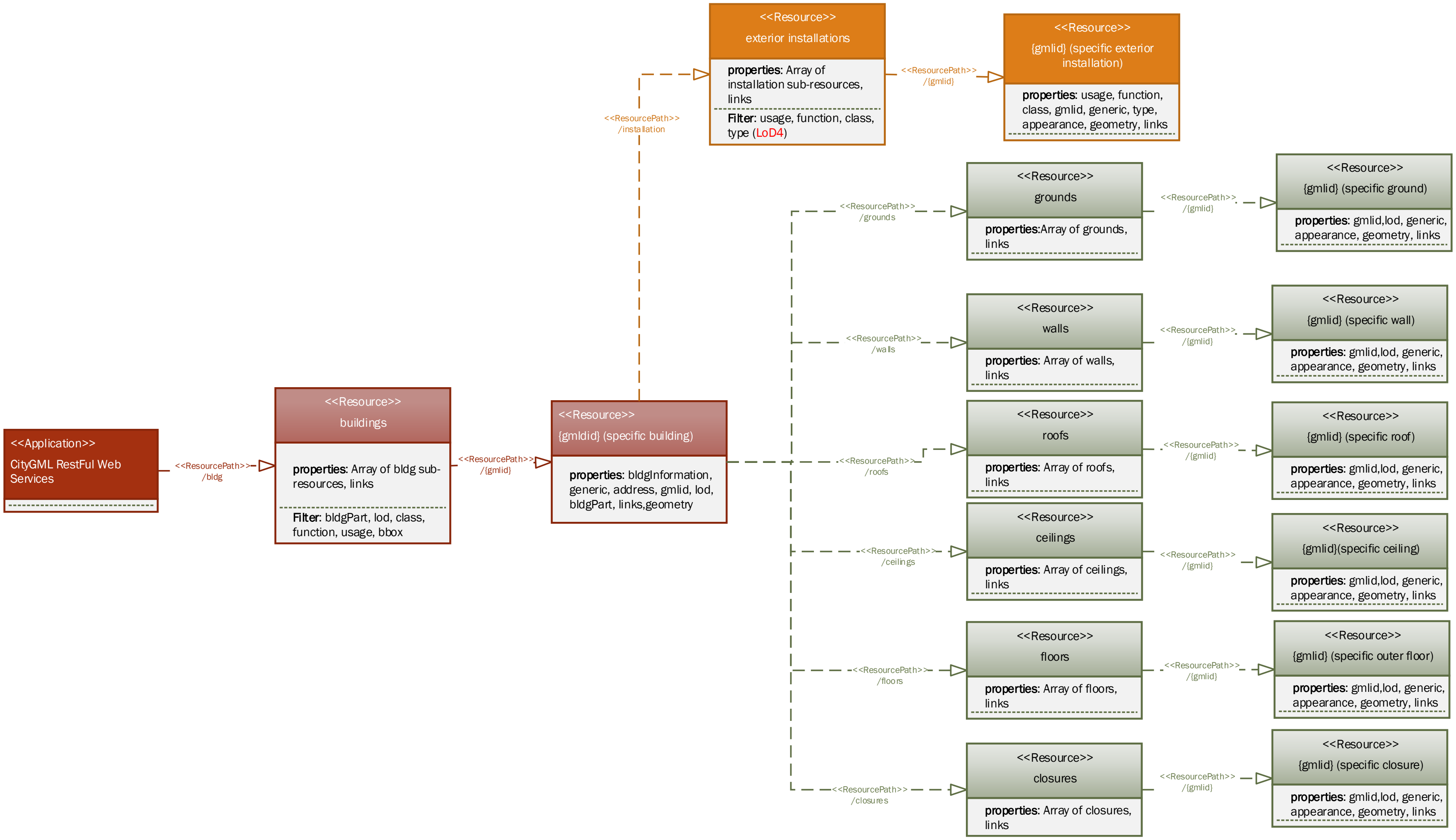
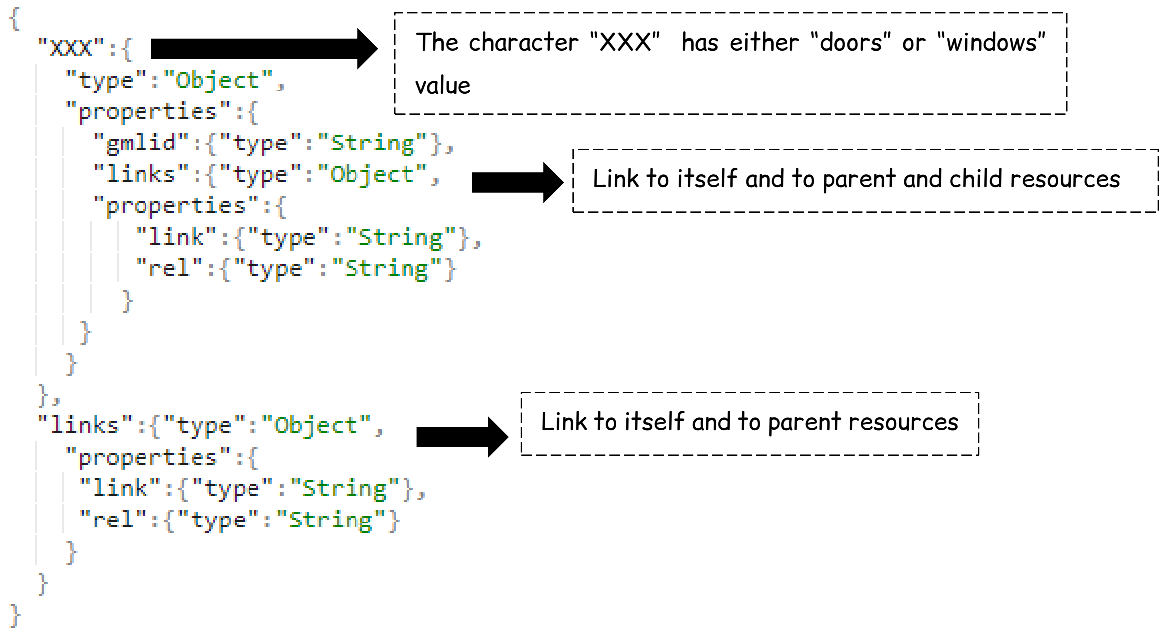

<<Application>>
CityGML RestFul Web Services
<<ResourcePath>>
/bldg
<<Resource>>
buildings
properties: Array of bldg sub-resources, links
Filter: bldgPart, lod, class, function, usage, bbox
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmldid} (specific building)
properties: bldgInformation, generic, address, gmlid, lod, bldgPart, links,geometry
<<ResourcePath>>
/installation
<<Resource>>
exterior installations
properties: Array of installation sub-resources, links
Filter: usage, function, class, type (LoD4)
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific exterior installation)
properties: usage, function, class, gmlid, generic, type, appearance, geometry, links
<<ResourcePath>>
/grounds
<<Resource>>
grounds
properties:Array of grounds, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific ground)
properties: gmlid,lod, generic, appearance, geometry, links
<<ResourcePath>>
/walls
<<Resource>>
walls
properties: Array of walls, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific wall)
properties: gmlid,lod, generic, appearance, geometry, links
<<ResourcePath>>
/roofs
<<Resource>>
roofs
properties: Array of roofs, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific roof)
properties: gmlid,lod, generic, appearance, geometry, links
<<ResourcePath>>
/ceilings
<<Resource>>
ceilings
properties: Array of ceilings, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid}(specific ceiling)
properties: gmlid,lod, generic, appearance, geometry, links
<<ResourcePath>>
/floors
<<Resource>>
floors
properties: Array of floors, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific outer floor)
properties: gmlid,lod, generic, appearance, geometry, links
<<ResourcePath>>
/closures
<<Resource>>
closures
properties: Array of closures, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific closure)
properties: gmlid,lod, generic, appearance, geometry, links

Figure 4-12: Conceptual design of the LoD2 "bldg." sub-resources

CHAPTER 4: LOD-BASED THEMATIC RESOURCES

4.1.3. LoD3 bldg sub-resources

The additional semantic feature of the LoD3 building module is the "_Opening" abstract class, which semantically describes openings like doors and windows in outer or inner boundary surfaces like walls and roofs. This class only exists in models of LoD3 and LoD4 and contains two sub-classes such as "Window" and "Door". More specific, the class "Window" is used for modelling windows in the exterior shell of a building, or hatches between adjacent rooms, whist the "Door" class is used for modelling doors in the exterior shell of a building, or between adjacent rooms. The main difference between these classes is that the "Window" class is not specifically intended for the transit of people or vehicles and the "Door" can also be used by people to enter or leave a building or room (Gröger et al., 2012). Consequently, the respective resources of the aforementioned sub-classes are considered vital to be defined. Hence, the URIs of these resources are "windows" and "doors". The implementation of these sub-resources retrieves two objects such as "rooms" or "windows" (depend on the resource) and "links", which are presented in the following JSON-based schema (Figure 4-13) (for more details see Annex A.3).



```
{
  "XXX":{
    "type":"Object",
    "properties":{
      "gmlid":{"type":"String"},
      "links":{"type":"Object",
      "properties":{
         "link":{"type":"String"},
         "rel":{"type":"String"}
         }
      }
    }
  },
  "links":{"type":"Object",
    "properties":{
     "link":{"type":"String"},
     "rel":{"type":"String"}
    }
  }
}
```

The character "XXX" has either "doors" or "windows" value

Link to itself and to parent and child resources

Link to itself and to parent resources

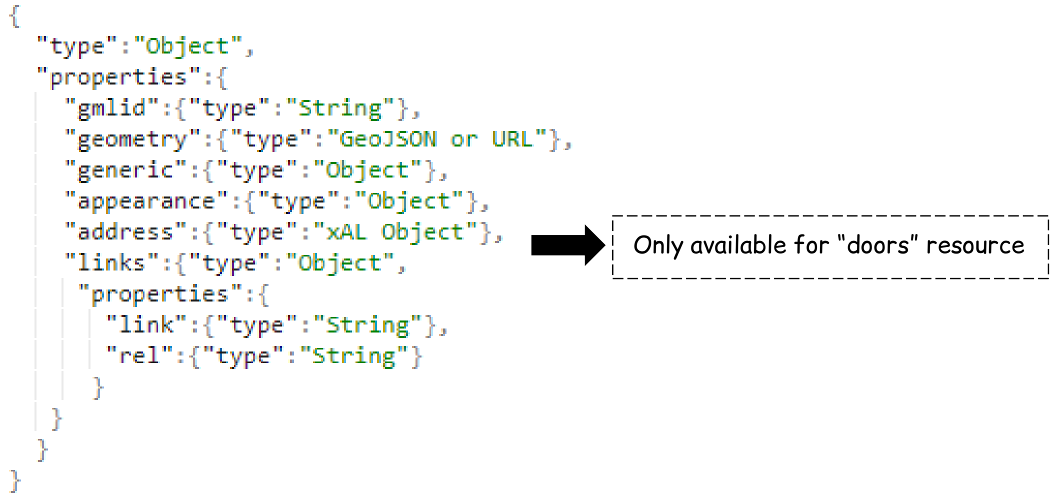
Figure 4-13: JSON-based schema of the "doors" and "windows" sub-resources

The implementation of a specific opening sub-resource contains a variety of information which is described in Table 4-3 and presented in Figure 4-14.

Information	Type	Description
gmlid	String	Gmlid value
appearance	Object	List of key value pairs based on appearance module
geometry	Object	Geometry object based on GeoJSON specification or URL to external format.
generic	Object	Ad hoc list of key value pairs based on generic module
links	Object	List of key value pairs regarding links to itself and to parent resources
address*	Object	List of key value pairs based on xAL specification

address: available only for "doors" sub-resources*

Table 4-3: Available information of "windows" and "doors" sub-resources



```
{
  "type":"Object",
  "properties":{
    "gmlid":{"type":"String"},
    "geometry":{"type":"GeoJSON or URL"},
    "generic":{"type":"Object"},
    "appearance":{"type":"Object"},
    "address":{"type":"xAL Object"},
    "links":{"type":"Object",
     "properties":{
       "link":{"type":"String"},
       "rel":{"type":"String"}
     }
    }
  }
}
```

Only available for "doors" resource

Figure 4-14: JSON-based resource schema of specific "opening" resource

Additionally, the schema of the URI regarding a specific opening resource is the following:

../bldg/{gmlid}/{roofs or walls}/{gmlid}/{openings}/{gmlid}

The conceptual design of the bldg child resources regarding LoD3 is shown in Figure 4-15

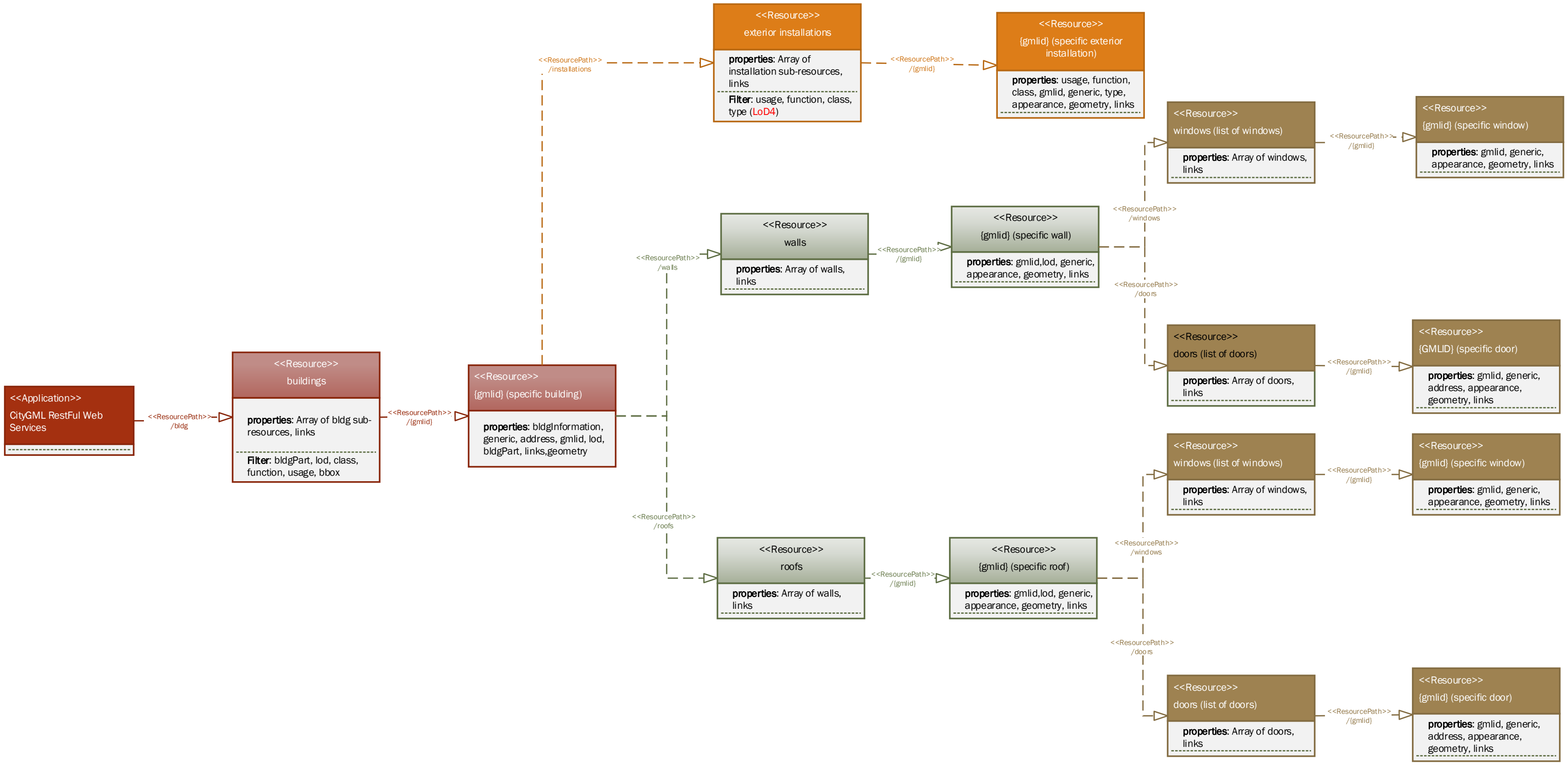

<<Application>>
CityGML RestFul Web Services
<<ResourcePath>>
/bldg
<<Resource>>
buildings
properties: Array of bldg sub-resources, links
Filter: bldgPart, lod, class, function, usage, bbox
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific building)
properties: bldgInformation, generic, address, gmlid, lod, bldgPart, links,geometry
<<ResourcePath>>
/installations
<<Resource>>
exterior installations
properties: Array of installation sub-resources, links
Filter: usage, function, class, type (LoD4)
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific exterior installation)
properties: usage, function, class, gmlid, generic, type, appearance, geometry, links
<<ResourcePath>>
/walls
<<Resource>>
walls
properties: Array of walls, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific wall)
properties: gmlid,lod, generic, appearance, geometry, links
<<ResourcePath>>
/windows
<<Resource>>
windows (list of windows)
properties: Array of windows, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific window)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>>
/doors
<<Resource>>
doors (list of doors)
properties: Array of doors, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{GMLID} (specific door)
properties: gmlid, generic, address, appearance, geometry, links
<<ResourcePath>>
/roofs
<<Resource>>
roofs
properties: Array of walls, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific roof)
properties: gmlid,lod, generic, appearance, geometry, links
<<ResourcePath>>
/windows
<<Resource>>
windows (list of windows)
properties: Array of windows, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific window)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>>
/doors
<<Resource>>
doors (list of doors)
properties: Array of doors, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific door)
properties: gmlid, generic, address, appearance, geometry, links

Figure 4-15: Conceptual design of the LoD3 "bldg." sub-resources

4.1.4. LoD4 bldg sub-resources

In LoD4, the interior building installations could be retrieved by using the "installation" resource (see Annex A.2). These features are objects inside a building with a specialized function or semantic meaning and they are permanently attached to the building structure and cannot be moved. Except for this resource, there is the "rooms" child resource regarding LoD4 as well (see Annex A.4). According to the CityGML v2 specification (Gröger et al., 2012), a "Room" is a semantic object for modelling the free space inside a building and should be uniquely related to exactly one building or building part object. Therefore, the "rooms" resource could be used to retrieve the list of the available rooms of a building.

../bldg/{gmlid}/rooms	list of rooms
⬇	
../bldg/{gmlid}/rooms/{gmlid}	specific room

Moreover, the available information of each room is gmlid, class, usage, function, links and generic and the filtering of this resource could be achieved by implementing the general filters (see Table 3-4). Thereafter, each room provides several links for child resources such as "furniture" (see Annex A.5), "installation" (see Annex A.2), "walls", "floors" and "ceilings" (see Annex A.1). The first one retrieves a list of "BuildingFurniture" that are located in a specific room. A "BuildingFurniture" is a movable part of a room, such as a chair or furniture. Also, it should be uniquely related to exactly one room. So, the accessible information of the "furniture" resource is class, usage, function, gmlid, generic, appearance, geometry and links. Additionally, the available filter parameters of this resource are class, usage, function and bbox. In the same context, the rest of the child resources such as "installation", "walls", "floors" and "ceilings" retrieve a list of the corresponding available semantic features. The accessible retrieval information and the respective filters are shown in Figure 4-16. Generally, the retrieval of a specific semantic feature is achieved using the gmlid.

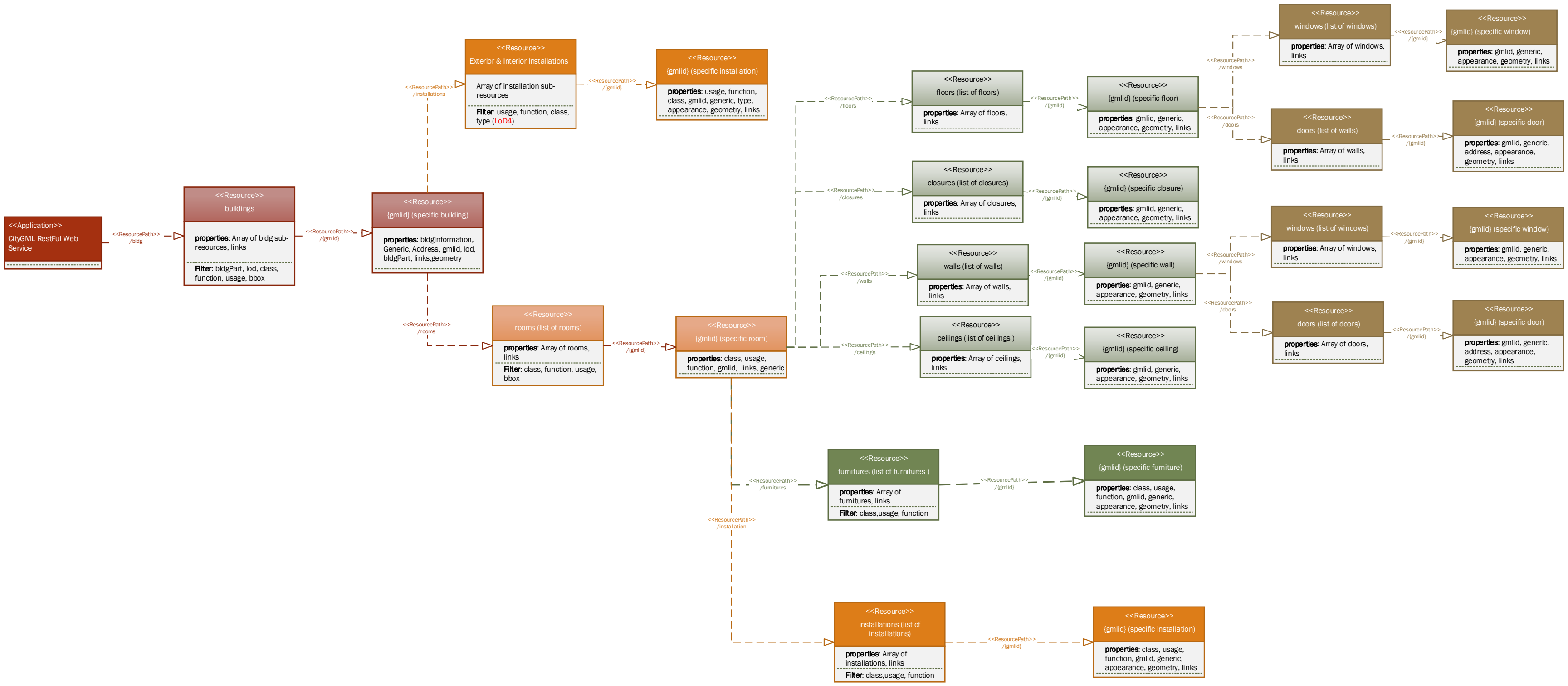

<<Application>>
CityGML RestFul Web Service
<<ResourcePath>>
/bldg
<<Resource>>
buildings
properties: Array of bldg sub-resources, links
Filter: bldgPart, lod, class, function, usage, bbox
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific building)
properties: bldgInformation, Generic, Address, gmlid, lod, bldgPart, links,geometry
<<ResourcePath>>
/installations
<<Resource>>
Exterior & Interior Installations
Array of installation sub-resources
Filter: usage, function, class, type (LoD4)
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific installation)
properties: usage, function, class, gmlid, generic, type, appearance, geometry, links
<<ResourcePath>>
/rooms
<<Resource>>
rooms (list of rooms)
properties: Array of rooms, links
Filter: class, function, usage, bbox
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific room)
properties: class, usage, function, gmlid, links, generic
<<ResourcePath>>
/floors
<<Resource>>
floors (list of floors)
properties: Array of floors, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific floor)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>>
/windows
<<Resource>>
windows (list of windows)
properties: Array of windows, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific window)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>>
/doors
<<Resource>>
doors (list of walls)
properties: Array of walls, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific door)
properties: gmlid, generic, address, appearance, geometry, links
<<ResourcePath>>
/closures
<<Resource>>
closures (list of closures)
properties: Array of closures, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific closure)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>>
/walls
<<Resource>>
walls (list of walls)
properties: Array of walls, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific wall)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>>
/windows
<<Resource>>
windows (list of windows)
properties: Array of windows, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific window)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>>
/doors
<<Resource>>
doors (list of doors)
properties: Array of doors, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific door)
properties: gmlid, generic, address, appearance, geometry, links
<<ResourcePath>>
/ceilings
<<Resource>>
ceilings (list of ceilings)
properties: Array of ceilings, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific ceiling)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>>
/furnitures
<<Resource>>
furnitures (list of furnitures)
properties: Array of furnitures, links
Filter: class,usage, function
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific furniture)
properties: class, usage, function, gmlid, generic, appearance, geometry, links
<<ResourcePath>>
/installation
<<Resource>>
installations (list of installations)
properties: Array of installations, links
Filter: class,usage, function
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific installation)
properties: class, usage, function, gmlid, generic, appearance, geometry, links

Figure 4-16: Conceptual design of the LoD4 "bldg." sub-resources

It should be noted that in LoD4 there are two sub-resources with the same name but different URIs, and called "installation". The first one is child resource of bldg resource and retrieve a list of interior installations in a particular building (1), while the second one is the child resource of the "rooms" resource and retrieve the respective installation that are located in a specific room (2) (see Annex A.2).

```
../bldg/{gmlid}/installation                                    (1)
```

```
../bldg/{gmlid}/rooms/{gmlid}/installation                      (2)
```

Similar to the LoD3, the interior boundary resources ("walls and "floors") provide the "windows" and "doors" child resources.

```
../bldg/{gmlid}/rooms/{gmlid}/{walls or floors}/{gmlid}/{windows or doors}
```

The aforesaid resources have similar properties, filters (see Table 4-3) and schema (see Annex A.3) like LoD3 opening resources.

4.1.5. Case studies using semantic requests

In this section, several requests are presented using the conceptual design of the CityGML RESTful Web service. For this purpose, the "Topo3DcityDBPS" 2D/3D WebGIS is utilized which was developed and presented by Pispidikis & Dimopoulou (2016) in order to successfully retrieve and visualize CityGML data in accordance with their respective geometric and semantic characteristics. Thereafter, Athanasiou, Pispidikis, & Dimopoulou (2018), for interoperability purposes, upgraded this application by replacing its main retrieval mechanism with the bldg resources of CityGML RESTful Web service.

Initially, a building example was chosen, which includes a variety of semantic characteristics in all LoDs (Figure 4-17)

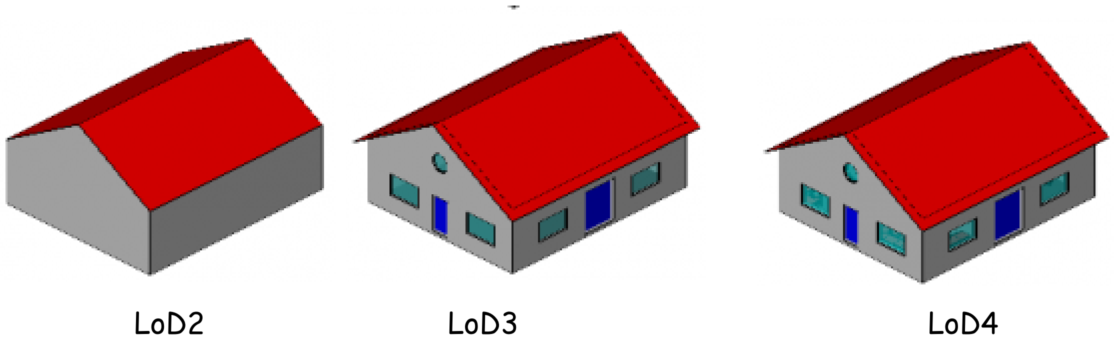
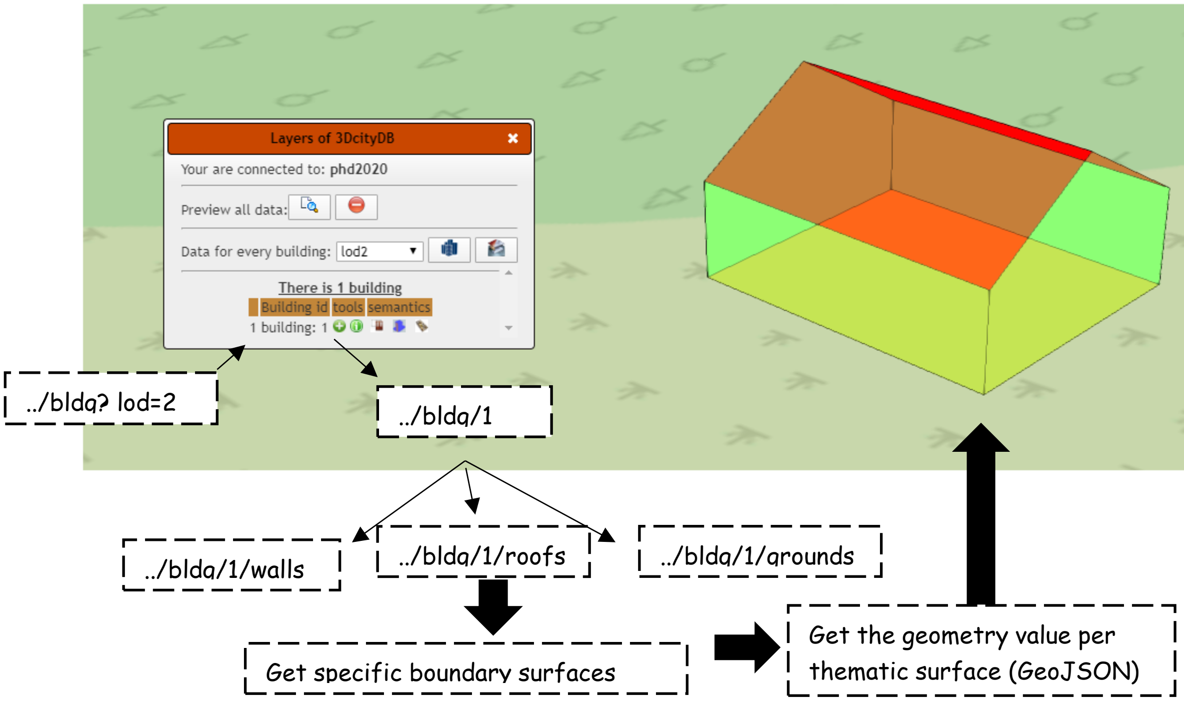


Figure 4-17: A building example in LoD2, LoD3 and LoD4

Next, a PostgreSQL/PostGIS Database was utilized which was structured according to 3DcityDB schema. Thereafter, the storage of this building model into this spatial database was implemented by the use of the 3DcityDB importer/exporter.

When the connection to the database was achieved, the available buildings in LoD2 were retrieved. Next, the building (id:1) was retrieved and visualized (Figure 4-18)


Layers of 3DcityDB

Your are connected to: phd2020

Preview all data:

Data for every building: lod2

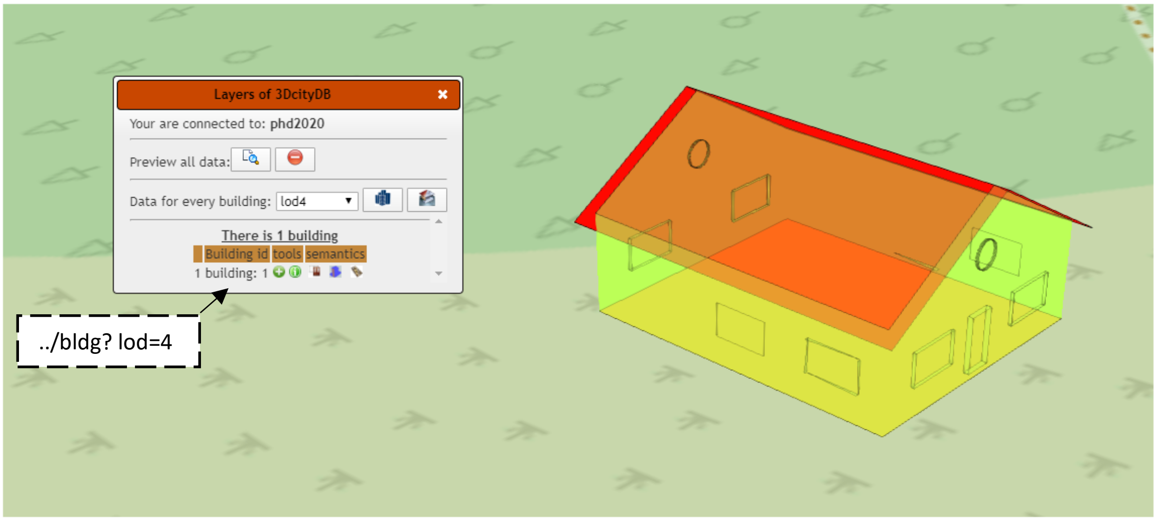
There is 1 building

Building id	tools	semantics
1 building: 1		

../bldg? lod=2
../bldg/1
../bldg/1/walls
../bldg/1/roofs
../bldg/1/grounds
Get specific boundary surfaces
Get the geometry value per thematic surface (GeoJSON)

Figure 4-18: LoD2 bldg sub-resources implementation example

Similarly, Figure 4-19 presents the retrieval of LoD4 instance of this building



Your are connected to: phd2020

Data for every building: lod4

There is 1 building

Building id	tools	semantics

1 building: 1

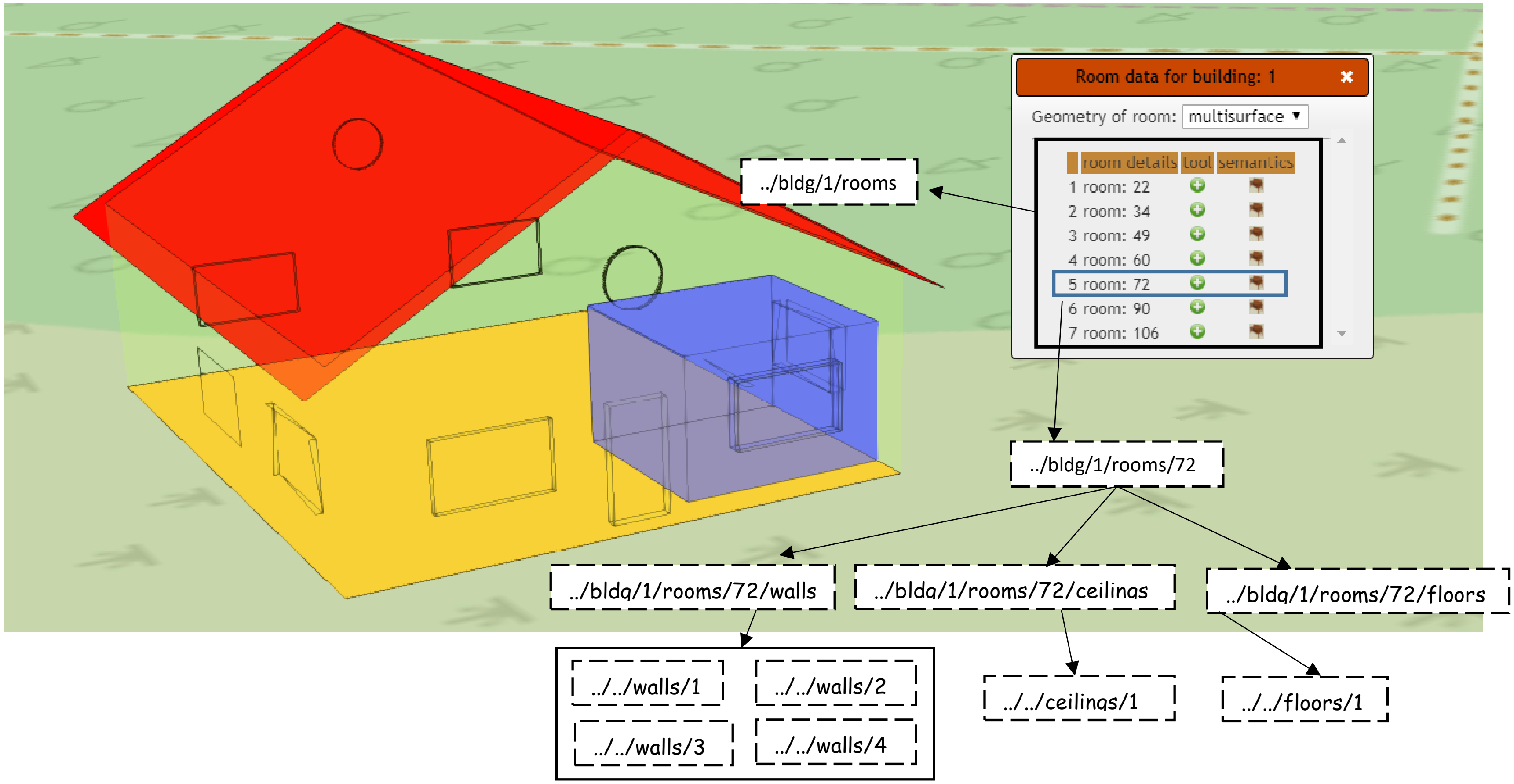
../bldg? lod=4

Figure 4-19: LoD4 bldg sub-resources implementation example

Next, the following request is used to fetch all the available rooms of this building:

```
../bldg/1/rooms   → seven rooms
```

Thereafter, having all the available rooms (seven rooms in total) the retrieval of a particular room (e.g. id: 72) is implemented and visualized as follows (Figure 4-20):



Room data for building: 1 ✖

Geometry of room: multisurface ▾

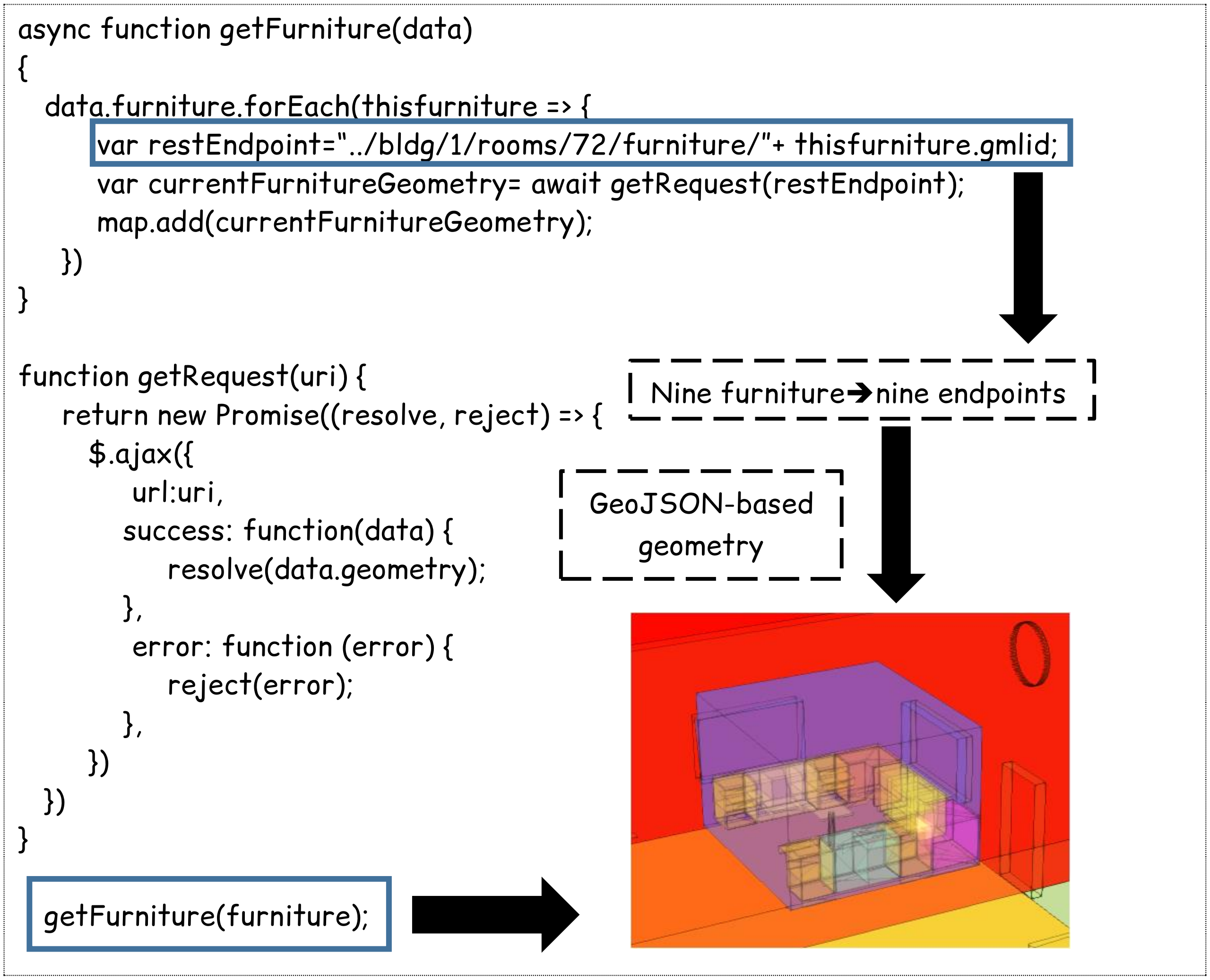
	room details	tool	semantics
1	room: 22	⊕	●
2	room: 34	⊕	●
3	room: 49	⊕	●
4	room: 60	⊕	●
5	room: 72	⊕	●
6	room: 90	⊕	●
7	room: 106	⊕	●

Figure 4-20: Example of "rooms" resources implementation

Finally, the retrieval and visualization of all the available furniture of the room 72 is achieved by implementing the following procedure. Firstly, these furniture are retrieved using the following endpoint:

```
../bldg/1/rooms/72/furniture                                1st request
```

Next, the said result is implemented as JSON input in JavaScript code (Figure 4-21):



```
async function getFurniture(data)
{
   data.furniture.forEach(thisfurniture => {
        var restEndpoint="../bldg/1/rooms/72/furniture/"+ thisfurniture.gmlid;
        var currentFurnitureGeometry= await getRequest(restEndpoint);
        map.add(currentFurnitureGeometry);
     })
}

function getRequest(uri) {
     return new Promise((resolve, reject) => {
        $.ajax({
             url:uri,
           success: function(data) {
                resolve(data.geometry);
           },
            error: function (error) {
                reject(error);
           },
        })
   })
}

getFurniture(furniture);
```

Figure 4-21: Advanced requests to fetch all the furniture in a specific room

4.2. Tun Thematic Resource

The tunnel model is closely related to the building model. The scope of this model encompasses manmade structures that are located mostly below the terrain surface and are intended to convey transportation flows such as pedestrians, cars, trains etc. Therefore, the geological structures, natural caves, mining facilities and subsurface utility network are excluded (Gröger et al., 2012). Additionally, this model supports the representation of semantic aspects of tunnel and tunnel parts only in four levels of detail, LoD1 to LoD4 (Figure 4-22).

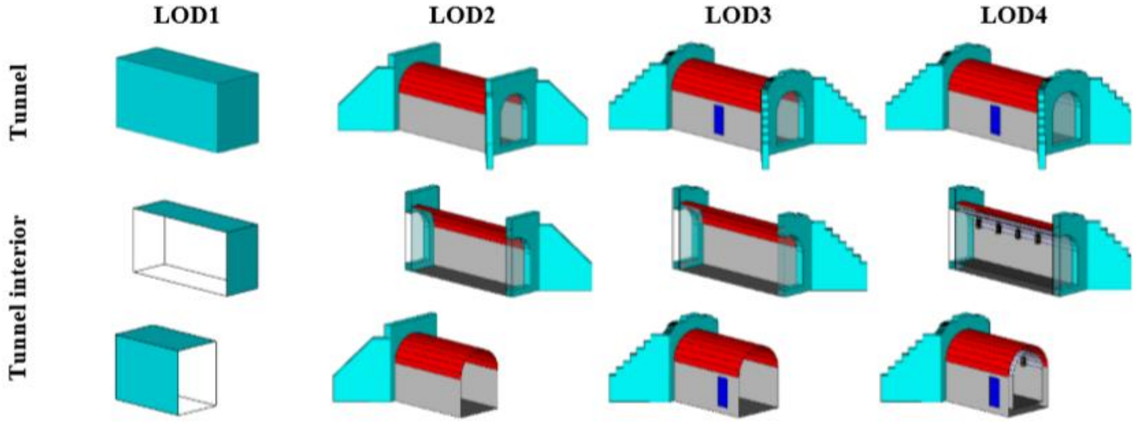


Figure 4-22: Tunnel module in different LoDs

(Gröger et al., 2012)

Specifically, in LoD1, there are no semantic characteristics as the tunnel model consists only of a geometric representation of the tunnel volume. In LoD2 and higher LoDs the outer structure of a tunnel can be semantically differentiated while in LoD4, the interior of a tunnel can also be structured with additional semantic features (Table 4-4).

	Geometric/ semantic theme	LoD1	LoD2	LoD3	LoD4
	Volume part of the tunnel shell	■	■	■	■
	Tunnel parts	■	■	■	■
Semantic themes	Boundary surfaces		■	■	■
	Outer tunnel installations		■	■	■
	Openings			■	■
	Hollow spaces				■
	Interior tunnel installation				■

Table 4-4: Semantic availability of a tunnel model per LoD

4.2.1. Tun main resource

The "tun" main resource refers to the tunnel module of the CityGML and is used to retrieve all the available tunnels and tunnel parts respectively. When a tunnel composed of structural segments, for example tunnel entrance and subway, has to be separated into one tunnel having one or more additional "TunnelPart" (Figure 4-23).

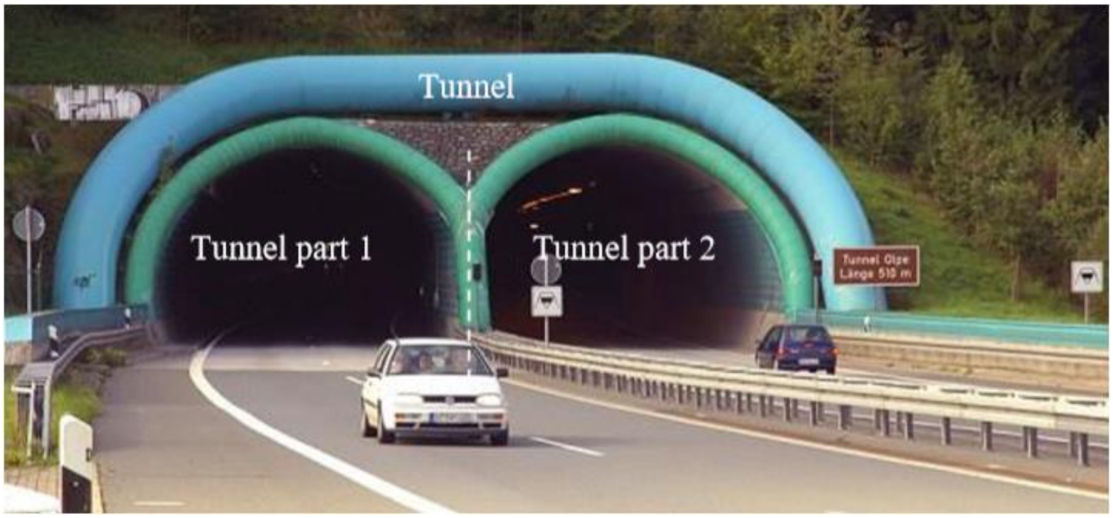
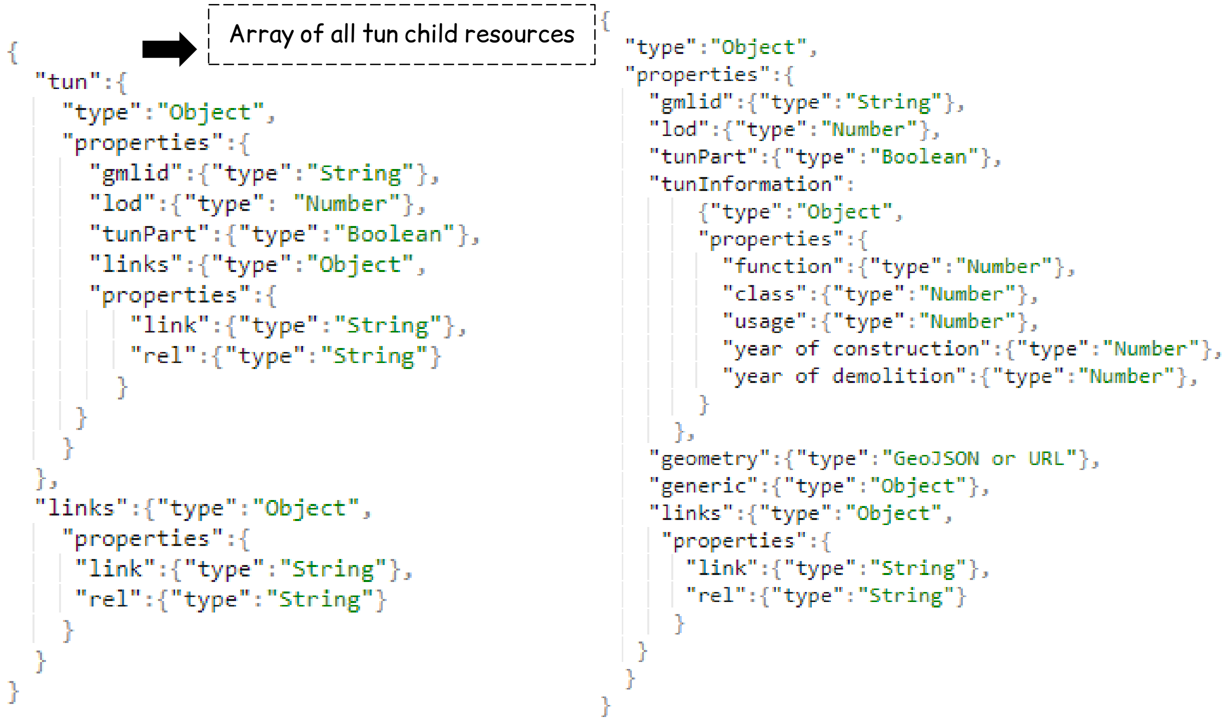


Figure 4-23: Example of a tunnel modeled with two tunnel parts

(Gröger et al., 2012)

The available filters for this resource are the general filters (see Table 3-4) and also the tunPart. The value of the latter is Boolean and describe whether the tunnel is tunnel part or not.

The retrieval resource schema contains two objects like "links" and "tun". The first one contains an array of links to itself and to parent resource ("citymodels" resource), while the second one contains the available tunnels and tunnel parts (Figure 4-24(a)).



```
{
  "tun":{
    "type":"Object",
    "properties":{
      "gmlid":{"type":"String"},
      "lod":{"type": "Number"},
      "tunPart":{"type":"Boolean"},
      "links":{"type":"Object",
      "properties":{
         "link":{"type":"String"},
         "rel":{"type":"String"}
         }
      }
    }
  },
  "links":{"type":"Object",
    "properties":{
     "link":{"type":"String"},
     "rel":{"type":"String"}
    }
  }
}
```

a) .../tun

```
{
 "type":"Object",
 "properties":{
   "gmlid":{"type":"String"},
   "lod":{"type":"Number"},
   "tunPart":{"type":"Boolean"},
   "tunInformation":
       {"type":"Object",
       "properties":{
         "function":{"type":"Number"},
         "class":{"type":"Number"},
         "usage":{"type":"Number"},
         "year of construction":{"type":"Number"},
         "year of demolition":{"type":"Number"},
         }
     },
   "geometry":{"type":"GeoJSON or URL"},
   "generic":{"type":"Object"},
   "links":{"type":"Object",
    "properties":{
      "link":{"type":"String"},
      "rel":{"type":"String"}
     }
   }
  }
}
```

b) .../tun/{gmlid}

Figure 4-24: JSON-based tun resource schema

Additionally, a particular "tun" resource can be retrieved by using as sub-resource the corresponding gmlid. The available information of this sub-resource contains a variety of properties such as lod, tunPart, tunInformation, geometry, generic, gmlid and links (see Table 3-5; for JSON-based schema see Figure 4-24(b)). More specific, the "tunInformation" object includes a list of properties such as class, function, usage, year of construction and year of demolition.

4.2.2. LoD2 tun sub-resources

The LoD2 child resources of the "tun" resource are based on the respective classes' "_BoundarySurface" and TunnelInstallation of tunnel module of CityGML. Namely, the "_BoundarySurface" is the abstract class for several thematic classes, structuring the exterior shell of a tunnel as well as the visible surface of hollow spaces and both outer and interior tunnel installations. The thematic classification of tunnel surfaces with regard to the "_BoundarySurface" class is illustrated in Figure 4-25.

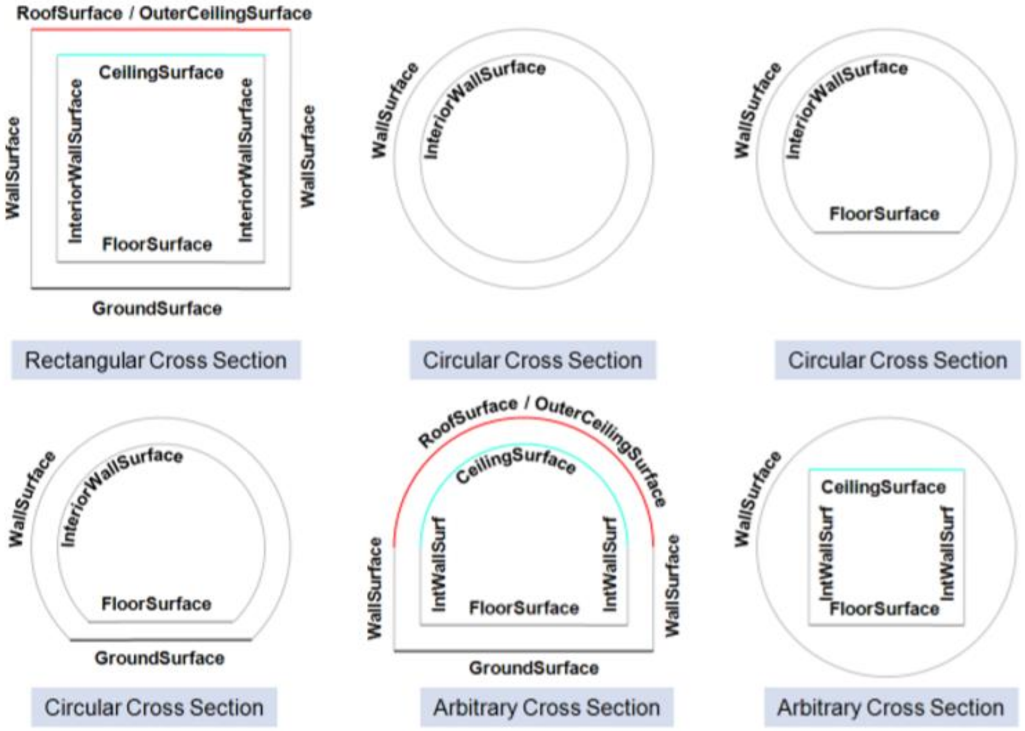
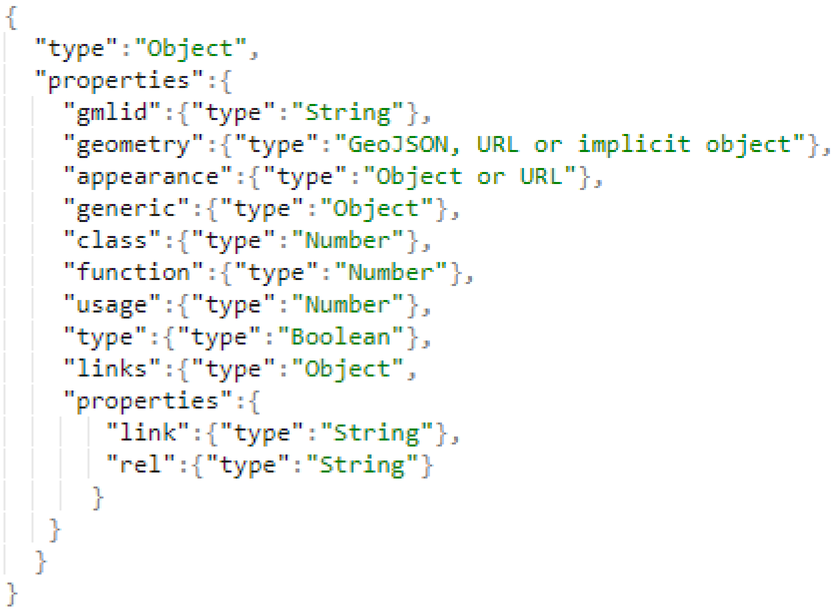


Figure 4-25: Exterior and interior tunnel boundary surfaces

(Gröger et al., 2012)

Therefore, in terms of the outer boundary surfaces, the following URIs are specified: "walls", "grounds", "roofs" and "ceilings". Additionally, the "closure" resource is also defined so that the open side of the model that was sealed by virtual surface can also be retrieved. The schema of the exterior boundary resources of tunnel is similar to that of the building resources either for the exterior boundary surface category (see Annex A.1) or for the features belonging to a specific exterior boundary surface (see Figure 4-8).

The semantic objects which refer to the outer components of a tunnel and strongly affect its outer characteristics belong to tunnel installations. So, the "installation" resource (see Annex A.2) is used for the retrieval of the aforementioned objects. This resource can be filtered by implementing a variety of filters such as class, function, usage and type. The property "type" is embedded to the "installation" resource so that the separation of the interior and exterior installation is achieved. The "installation" resource schema has a similar structure to exterior boundary surfaces apart from the fact that four attributes are included with respect to the retrieval resource schema of a particular installation such as class, function, usage and type (Figure 4-26).



```
{
  "type":"Object",
  "properties":{
    "gmlid":{"type":"String"},
    "geometry":{"type":"GeoJSON, URL or implicit object"},
    "appearance":{"type":"Object or URL"},
    "generic":{"type":"Object"},
    "class":{"type":"Number"},
    "function":{"type":"Number"},
    "usage":{"type":"Number"},
    "type":{"type":"Boolean"},
    "links":{"type":"Object",
    "properties":{
       "link":{"type":"String"},
       "rel":{"type":"String"}
      }
   }
  }
}
```

Figure 4-26: JSON-based retrieval resource schema of a specific tunnel installation

The conceptual design of the LoD2 "tun" resources is presented in Figure 4-27.

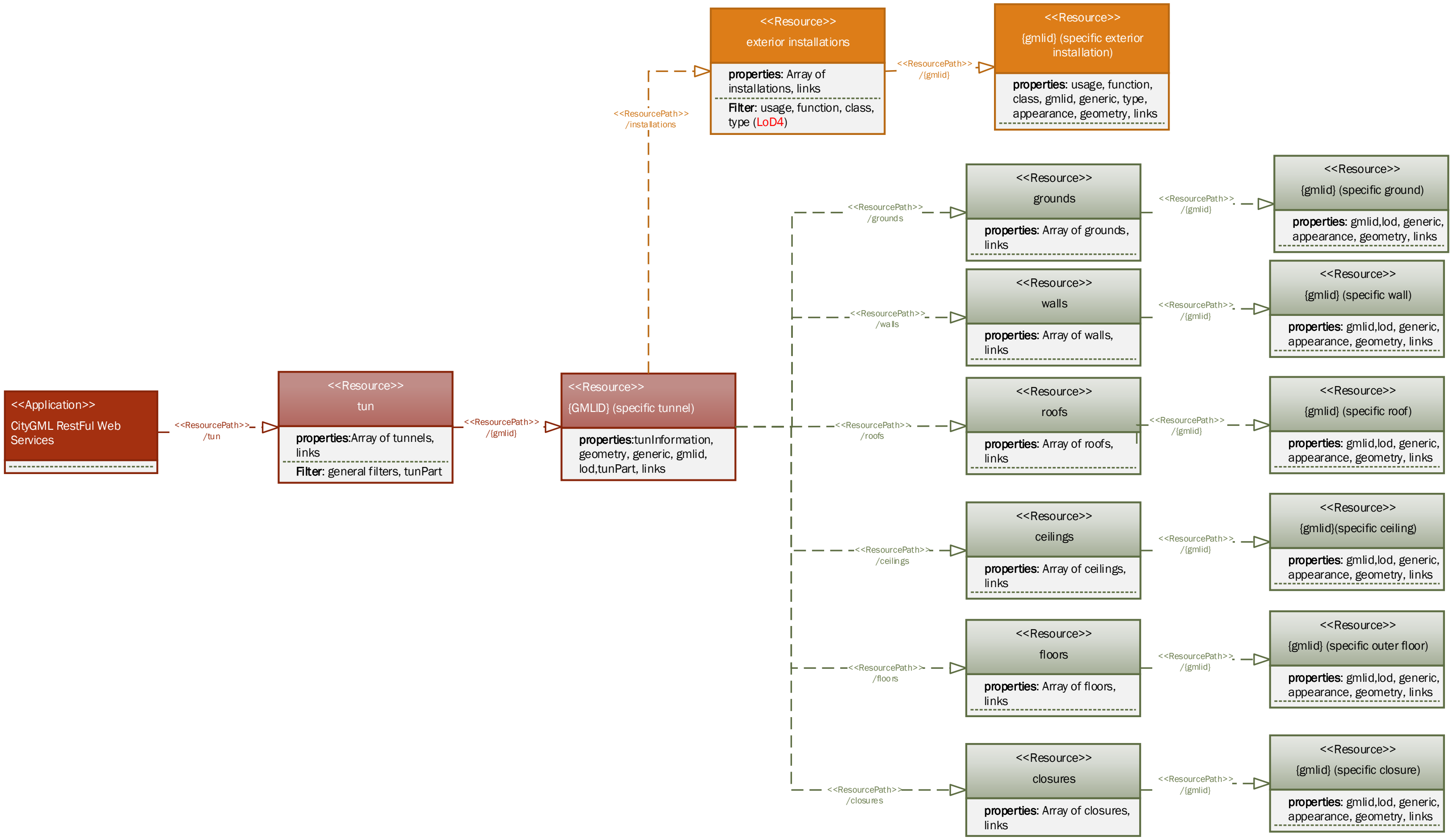

<<Application>>
CityGML RestFul Web Services
<<ResourcePath>> /tun
<<Resource>>
tun
properties:Array of tunnels, links
Filter: general filters, tunPart
<<ResourcePath>> /{gmlid}
<<Resource>>
{GMLID} (specific tunnel)
properties:tunInformation, geometry, generic, gmlid, lod,tunPart, links
<<ResourcePath>> /installations
<<Resource>>
exterior installations
properties: Array of installations, links
Filter: usage, function, class, type (LoD4)
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific exterior installation)
properties: usage, function, class, gmlid, generic, type, appearance, geometry, links
<<ResourcePath>> /grounds
<<Resource>>
grounds
properties: Array of grounds, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific ground)
properties: gmlid,lod, generic, appearance, geometry, links
<<ResourcePath>> /walls
<<Resource>>
walls
properties: Array of walls, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific wall)
properties: gmlid,lod, generic, appearance, geometry, links
<<ResourcePath>> /roofs
<<Resource>>
roofs
properties: Array of roofs, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific roof)
properties: gmlid,lod, generic, appearance, geometry, links
<<ResourcePath>> /ceilings
<<Resource>>
ceilings
properties: Array of ceilings, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid}(specific ceiling)
properties: gmlid,lod, generic, appearance, geometry, links
<<ResourcePath>> /floors
<<Resource>>
floors
properties: Array of floors, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific outer floor)
properties: gmlid,lod, generic, appearance, geometry, links
<<ResourcePath>> /closures
<<Resource>>
closures
properties: Array of closures, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific closure)
properties: gmlid,lod, generic, appearance, geometry, links

Figure 4-27: Conceptual design of the LoD2 "tun" sub-resources

4.2.3. LoD3 tun sub-resources

From LoD3 and above the exterior boundary surfaces such as roofs and walls may contain opening features like doors and windows. These features can be retrieved by using the "doors" and "windows" child resources respectively, which have similar information, filters and schema to the respective opening resources of building module (see Annex A.3). Additionally, the schema of the URI regarding a specific opening resource is the following:

```
../tun/{gmlid}/{roofs or walls}/{gmlid}/{openings}/{gmlid}
```

The conceptual design of the additional LoD3 tun sub-resources are presented in Figure 4-28

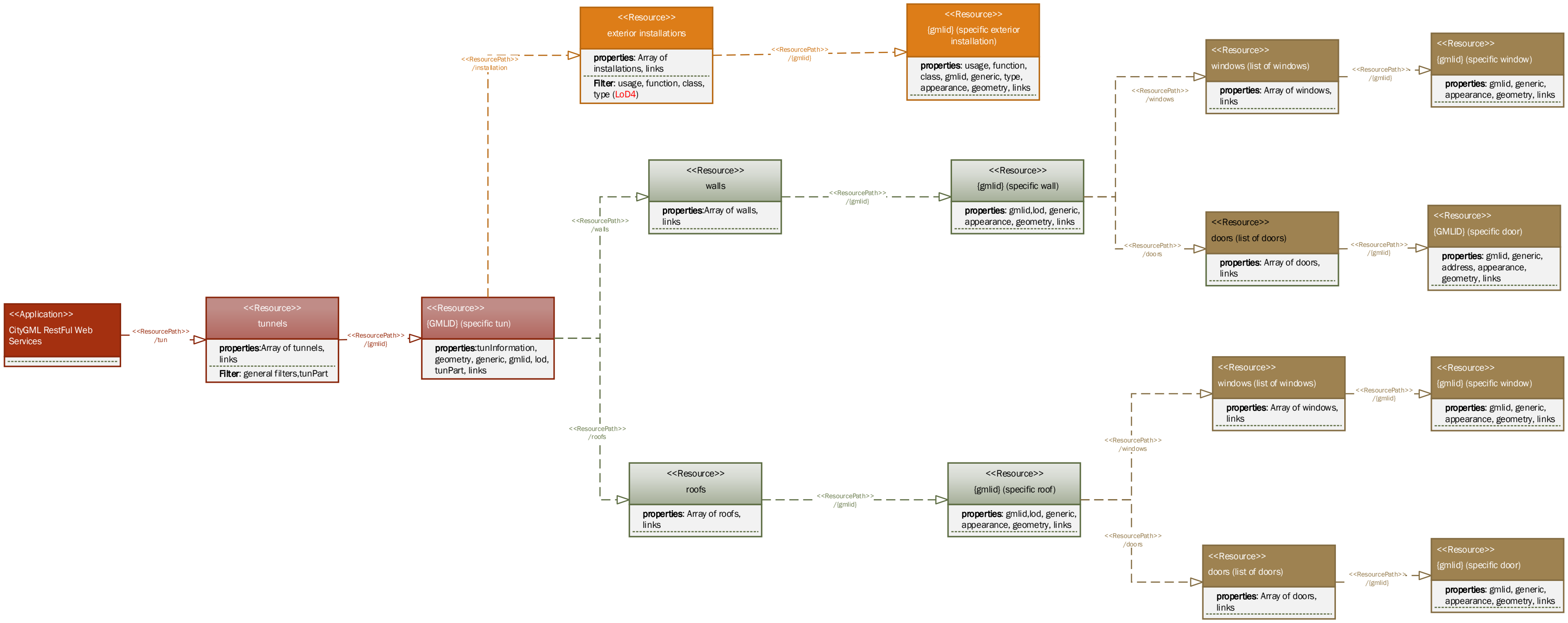

<<Application>>
CityGML RestFul Web Services
<<ResourcePath>> /tun
<<Resource>>
tunnels
properties:Array of tunnels, links
Filter: general filters,tunPart
<<ResourcePath>> /{gmlid}
<<Resource>>
{GMLID} (specific tun)
properties:tunInformation, geometry, generic, gmlid, lod, tunPart, links
<<ResourcePath>> /installation
<<Resource>>
exterior installations
properties: Array of installations, links
Filter: usage, function, class, type (LoD4)
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific exterior installation)
properties: usage, function, class, gmlid, generic, type, appearance, geometry, links
<<ResourcePath>> /walls
<<Resource>>
walls
properties:Array of walls, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific wall)
properties: gmlid,lod, generic, appearance, geometry, links
<<ResourcePath>> /windows
<<Resource>>
windows (list of windows)
properties: Array of windows, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific window)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>> /doors
<<Resource>>
doors (list of doors)
properties: Array of doors, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{GMLID} (specific door)
properties: gmlid, generic, address, appearance, geometry, links
<<ResourcePath>> /roofs
<<Resource>>
roofs
properties: Array of roofs, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific roof)
properties: gmlid,lod, generic, appearance, geometry, links
<<ResourcePath>> /windows
<<Resource>>
windows (list of windows)
properties: Array of windows, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific window)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>> /doors
<<Resource>>
doors (list of doors)
properties: Array of doors, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific door)
properties: gmlid, generic, appearance, geometry, links

Figure 4-28: Conceptual design of the additional LoD3 "tun" sub-resources

4.2.4. LoD4 tun sub-resources

In LoD4, the highest level of resolution, the interior of tunnel composed of several hollow spaces which mainly semantically describe the free space inside a tunnel or tunnel part. Therefore, the "hollowspaces" child resource regarding a specific "tun" resource can be requested and thus both the list of hollow spaces and the links objects can be retrieved. Thereafter, when a particular hollow space is requested by using the corresponding gmlid a variety of information is retrieved such as class, usage, function, gmlid, generic and links (see Annex A.6). Additionally, each hollow space can be semantically described and modeled by specialized boundary surfaces such as FloorSurface, CeilingSurface, InteriorWallsurface and ClosureSurface. Therefore, each "hollowspace" resource provides several links to the respective boundary child resources such as "walls", "floors", "ceilings" and "closures" (see Annex A.1). Then, a specific "walls" resource may provide as child resources the opening features such as windows and doors (see Annex A.3). Moreover, the objects inside a tunnel which are permanently attached to the tunnel structure and cannot be moved can be requested by using the "installation" resource (see Annex A.2). It should be noted that there are two available "installation" sub-resources with different URIs so that the interior installation can be retrieved based on either a specific tunnel or a specific hollowspace. An instance of the above-mentioned cases is as follows

```
../tun/{gmlid}/hollowspaces/{gmlid}/installation
```

```
../tun/{gmlid}/installation?type=interior
```

Additionally, the retrieval of the movable objects of a hollow space can be requested by implementing the "furniture" sub-resource (see Annex A.5)

The conceptual design of the additional sub-resources for the LoD4 "tun" resource, including the respective available filters and properties per sub-resource, is shown in Figure 4-29

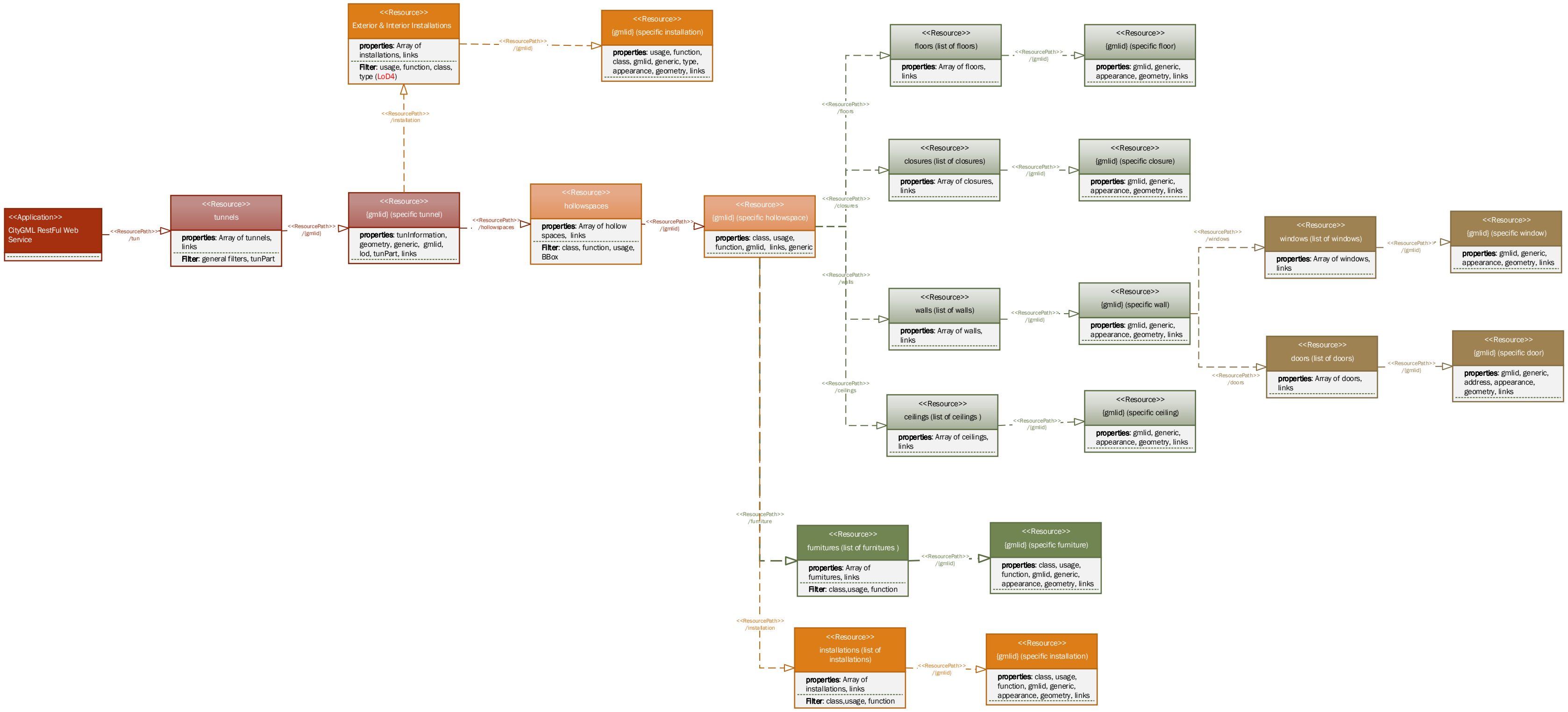

<<Application>>
CityGML RestFul Web Service
<<ResourcePath>>
/tun
<<Resource>>
tunnels
properties: Array of tunnels, links
Filter: general filters, tunPart
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific tunnel)
properties: tunInformation, geometry, generic, gmlid, lod, tunPart, links
<<ResourcePath>>
/installation
<<Resource>>
Exterior & Interior Installations
properties: Array of installations, links
Filter: usage, function, class, type (LoD4)
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific installation)
properties: usage, function, class, gmlid, generic, type, appearance, geometry, links
<<ResourcePath>>
/hollowspaces
<<Resource>>
hollowspaces
properties: Array of hollow spaces, links
Filter: class, function, usage, BBox
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific hollowspace)
properties: class, usage, function, gmlid, links, generic
<<ResourcePath>>
/floors
<<Resource>>
floors (list of floors)
properties: Array of floors, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific floor)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>>
/closures
<<Resource>>
closures (list of closures)
properties: Array of closures, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific closure)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>>
/walls
<<Resource>>
walls (list of walls)
properties: Array of walls, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific wall)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>>
/windows
<<Resource>>
windows (list of windows)
properties: Array of windows, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific window)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>>
/doors
<<Resource>>
doors (list of doors)
properties: Array of doors, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific door)
properties: gmlid, generic, address, appearance, geometry, links
<<ResourcePath>>
/ceilings
<<Resource>>
ceilings (list of ceilings)
properties: Array of ceilings, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific ceiling)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>>
/furniture
<<Resource>>
furnitures (list of furnitures)
properties: Array of furnitures, links
Filter: class,usage, function
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific furniture)
properties: class, usage, function, gmlid, generic, appearance, geometry, links
<<ResourcePath>>
/installation
<<Resource>>
installations (list of installations)
properties: Array of installations, links
Filter: class,usage, function
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific installation)
properties: class, usage, function, gmlid, generic, appearance, geometry, links

Figure 4-29: Conceptual design of the additional LoD4 "tun" sub-resources

4.2.5. Case studies using semantic requests

In this section, a variety of requests are presented using the conceptual design of the CityGML RESTful Web service regarding the "tun" resources. Initially, it should be noted that the code list values of function, usage and class with regard to semantic features such as tunnel, interior/exterior installations, hollow spaces and furniture are specified in the XML file CityGML_ExternalCodeList.xml, according to the dictionary concept of GML 3. Next, three main categories of requests are presented such as basic requests (a simple request), advanced requests (two or more requests) and requests using simple JavaScript code.

4.2.5.1. Basic requests

- Overview of the available tunnels and tunnel parts in LoD2.

```
../citymodels?thematics=tun&lod=2
```

- A CityGML dataset contains semantic information of tunnels in WGS84 CRS. However, a user needs to retrieve all the available pedestrian and roadway tunnels (functions: 1030 & 1010) in specific boundary area (334433.0, 4455667.0, 445677.0, 5566556.0) at Web Mercator Projection (EPSG: 3857).

```
../tun?function=1030,1010&bbox=334433.0, 4455667.0, 445677.0,
5566556.0&epsg=3857
```

- The exterior walls of a tunnel with gmlid 2.

```
../tun/2/walls
```

- The available windows of the wall with gmlid 2 for tunnel 1.

```
../tun/1/walls/2/windows
```

- The light switches (function: 3020, according to the interior tunnel installation dictionary) in tunnel 2.

```
../tun/2/installation?function=3020
```

- The lamps (function: 3010, according to the interior tunnel installation dictionary) of hollow space 3 for tunnel 2.

```
../tun/2/hollowspaces/3/installation?function=3010
```

- The furniture of hollow space 3 for tunnel 4.

```
../tun/4/hollowspaces/3/furniture
```

4.2.5.2. Advanced requests

Each HTTP request should happen in complete isolation (stateless interaction). Therefore, when the retrieval information is complex and needs more than one requests 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 constraints then the endpoint of every subsequent request can be retrieved from the "links" object of the current request.

- A CityGML dataset containing a tunnel with two tunnel parts in LoD2 (Figure 4-30). A user needs to retrieve all the lamps (function: 3010, according to the interior building installation dictionary) of hollowspaces for this tunnel.

-

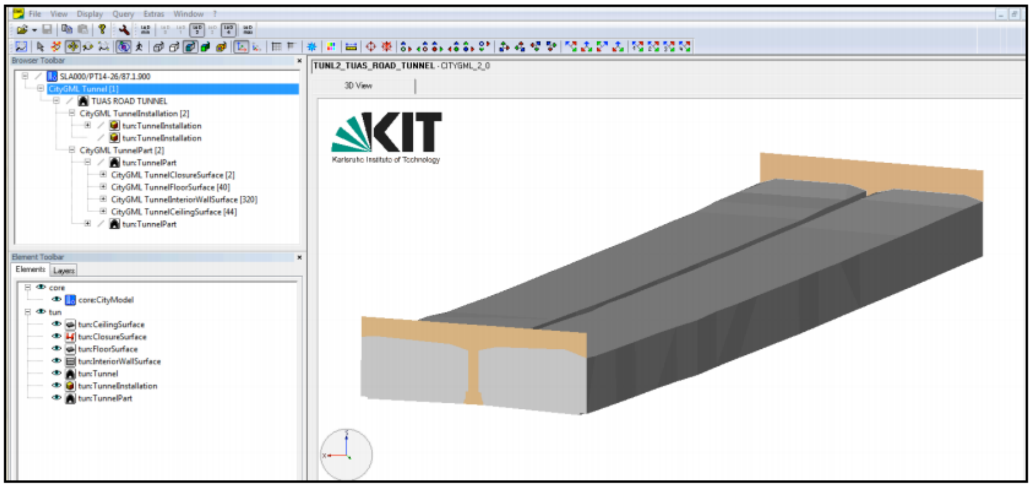


Figure 4-30: LoD2 Tunnel model

(Soon & Khoo, 2017)

-

So, in the first request, the user retrieves the available tunnel parts

```
../tun?tunPart=true              1st request➔ two tunnel parts (gmlid: tprt1 & tprt2)
```

For each tunnel parts the respective links are retrieved and thereafter, the available hollow spaces are retrieved as well.

```
../tun/trpt1/hollowspaces                         ../tun/trpt2/hollowspaces
```

⬇ One hollow space available (gmlid: hs1)

⬇ One hollow space available (gmlid: hs2)

Then, for each hollowspace the respective lamps are retrieved.

```
../tun/trpt1/hollowspaces/hs1/installation?function=3010
```

```
../tun/trpt2/hollowspaces/hs2/installation?function=3010
```

4.2.5.3. Requests using simple JavaScript code

- The number of burned out lamps in the hollow space (gmlid: hs1) for a particular tunnel (gmlid: tn2). 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: {type: "Boolean}
```

The first request is used to retrieve all the available lamps of the given hollow space implementing as sub-resource the "installation" resource in conjunctions with the respective filter regarding the installation function type.

```
../tun/tn2/hollowspaces/hs1/installation?function=3010
```

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

```
var count=0

 response.forEach(results => {
        if(results.installation.generic.burned==true) {
         count++;
         }
    })

console.log(count);   Result
```

4.3. Brid Thematic Resource

The bridge model represents the thematic, spatial and visual aspect of bridges, bridge parts and construction elements in four levels of detail (Figure 4-31) Additionally, it was also developed in strict analogy to the building model with respect to its aggregation structure, its relations, its attributes and the definition of the particular LoD (Gröger & Plümer, 2012).

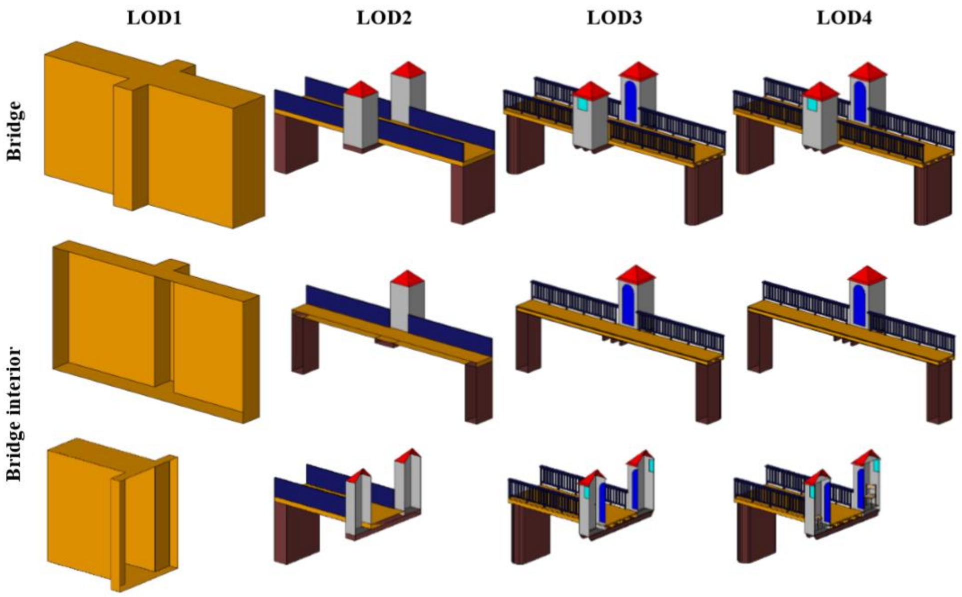


Figure 4-31: Bridge module in different LoD

(Gröger et al., 2012)

Therefore, the semantical and geometrical richness of the bridge module increases from LoD1 to LoD3 regarding the blocks and architectural model respectively, while the interior structures like rooms are embedded in LoD4 (Table 4-5)

	Geometric/ semantic theme	LoD1	LoD2	LoD3	LoD4
	Volume part of the bridge shell	■	■	■	■
	Bridge parts	■	■	■	■
Semantic themes	Bridge Construction elements	■	■	■	■
	Boundary surfaces		■	■	■
	Outer bridge installations		■	■	■
	Openings			■	■
	Rooms				■
	Interior bridge installation				■

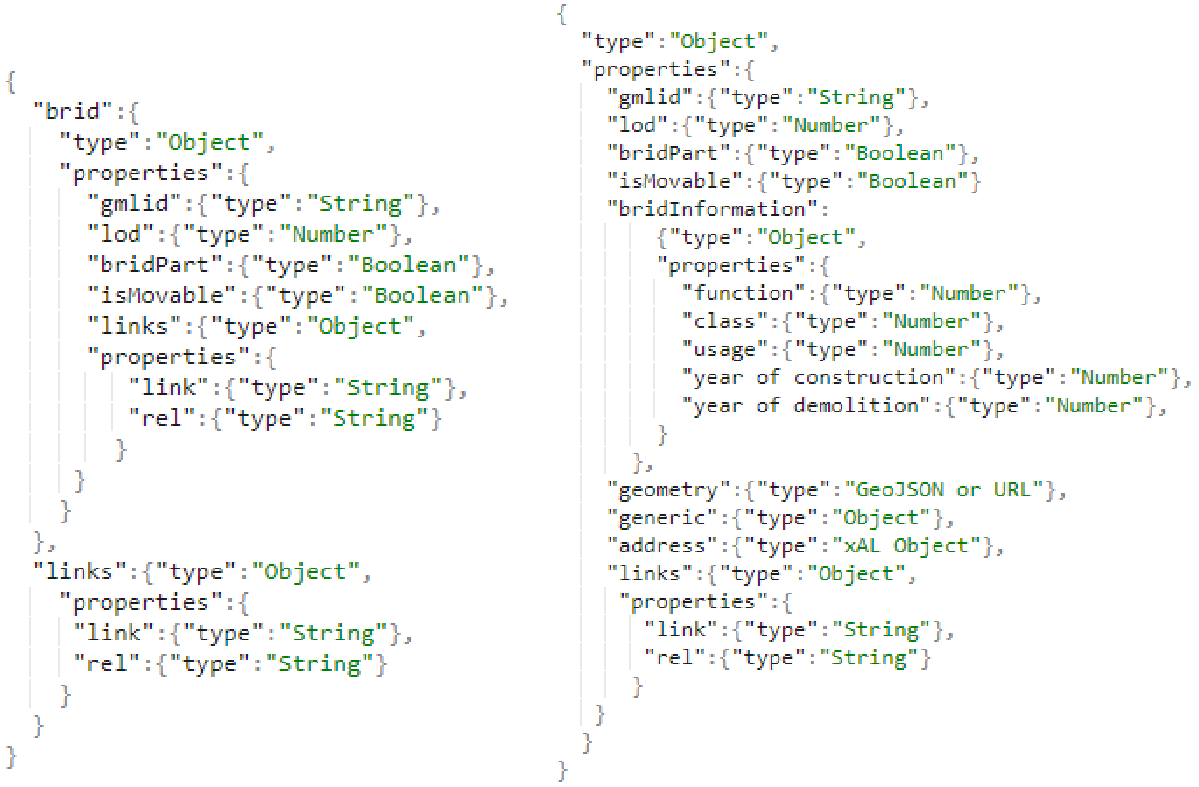
Table 4-5: Semantic and geometric availability of a bridge model per LoD

4.3.1. Brid main resource

The “brid” resource is the main thematic resource regarding the bridge module of the CityGML and it is used to retrieve a list of the available bridges. However, if some parts of a bridge differ from the remaining bridge regarding attribute values or if parts like ramps can be identified as objects, those parts can be represented as bridge parts and they can be retrieved as well. Additionally, the “links” object is also retrieved including information for links to itself and to “citymodels” resource URI (Figure 4-32(a)). Moreover, the retrieval list can be limited using not only the general filters (see Table 3-4) but also the “bridPart” and the “isMovable” filter parameters. The value of these filters is Boolean and provide information about whether the bridge is bridge part or not and whether is movable or immovable respectively.

The implementation of the gmlid attribute as sub-resource, it fetches information about a specific bridge or bridge part that contains a variety of properties (see Table 3-5; for JSON-based schema see Figure 4-32(b)).

The semantical richness of bridge module increases from LoD1 and above and thus, in LoD1, the semantic elements named “BridgeConstructionElement” are included. These features are considered essential from a structural point of view like pylons, anchorages etc. (Figure 4-33). Hence, the “construction” endpoint is defined as sub-resource. The information retrieval of said resource is an array of the available bridge construction elements and a list of links as well. Moreover, the array of the bridge construction elements can be filtered by using the general filters (see Table 3-4). Additionally, each of these elements can be retrieved using the corresponding gmlid as sub-resource (see Figure 4-34).



```
{
  "brid":{
    "type":"Object",
    "properties":{
      "gmlid":{"type":"String"},
      "lod":{"type":"Number"},
      "bridPart":{"type":"Boolean"},
      "isMovable":{"type":"Boolean"},
      "links":{"type":"Object",
      "properties":{
         "link":{"type":"String"},
         "rel":{"type":"String"}
        }
      }
    }
  },
  "links":{"type":"Object",
    "properties":{
     "link":{"type":"String"},
     "rel":{"type":"String"}
    }
  }
}
```

```
{
  "type":"Object",
  "properties":{
    "gmlid":{"type":"String"},
    "lod":{"type":"Number"},
    "bridPart":{"type":"Boolean"},
    "isMovable":{"type":"Boolean"}
    "bridInformation":
        {"type":"Object",
        "properties":{
          "function":{"type":"Number"},
          "class":{"type":"Number"},
          "usage":{"type":"Number"},
          "year of construction":{"type":"Number"},
          "year of demolition":{"type":"Number"},
        }
      },
    "geometry":{"type":"GeoJSON or URL"},
    "generic":{"type":"Object"},
    "address":{"type":"xAL Object"},
    "links":{"type":"Object",
     "properties":{
       "link":{"type":"String"},
       "rel":{"type":"String"}
      }
   }
  }
}
```

a) …/brid

b)…/brid/{gmlid}

Figure 4-32: JSON-based schema of a “brid” resource

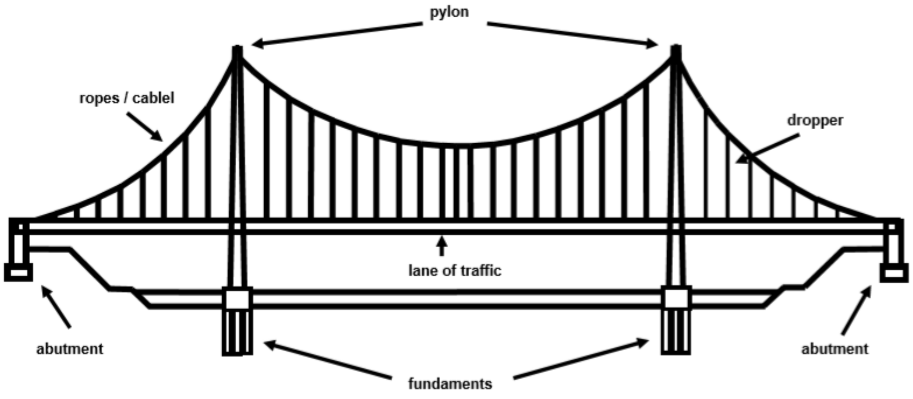
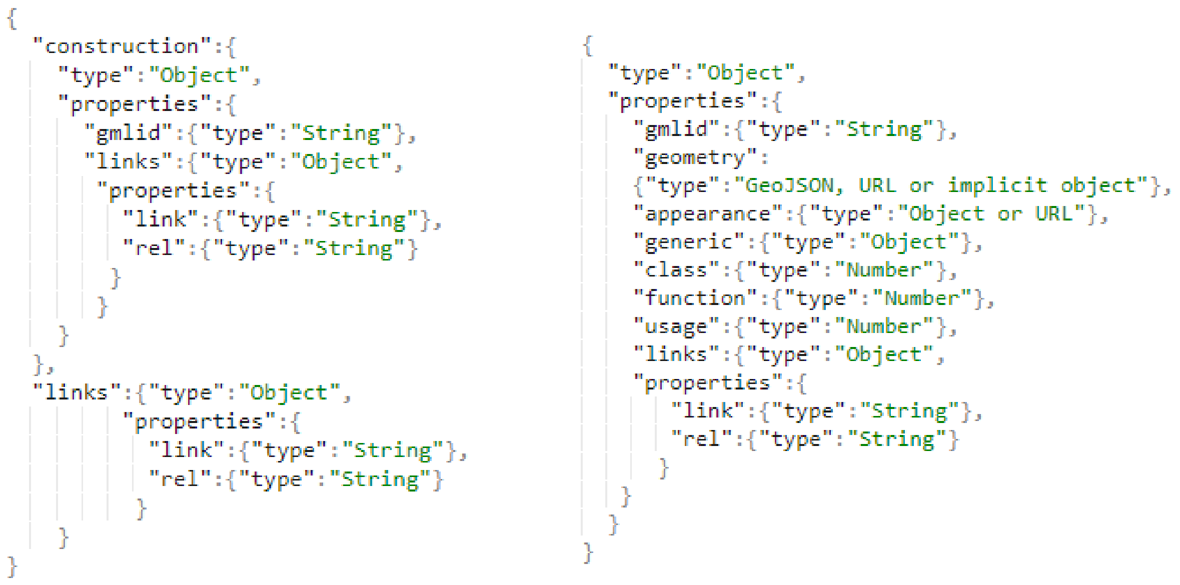


Figure 4-33: Bridge construction elements

(Gröger et al., 2012)



```
{
  "construction":{
    "type":"Object",
    "properties":{
      "gmlid":{"type":"String"},
      "links":{"type":"Object",
       "properties":{
         "link":{"type":"String"},
         "rel":{"type":"String"}
        }
       }
    }
  },
  "links":{"type":"Object",
         "properties":{
           "link":{"type":"String"},
           "rel":{"type":"String"}
          }
    }
}
```

```
{
  "type":"Object",
  "properties":{
    "gmlid":{"type":"String"},
    "geometry":
    {"type":"GeoJSON, URL or implicit object"},
    "appearance":{"type":"Object or URL"},
    "generic":{"type":"Object"},
    "class":{"type":"Number"},
    "function":{"type":"Number"},
    "usage":{"type":"Number"},
    "links":{"type":"Object",
    "properties":{
       "link":{"type":"String"},
       "rel":{"type":"String"}
      }
    }
   }
}
```

a) …/brid/{gmlid}/construction b) …/brid/{gmlid}/construction/{gmlid}

Figure 4-34: Construction resources of a bridge

4.3.2. LoD2 brid sub-resources

Except for the bridge construction elements, the additional semantic characteristics of the LoD2 bridge are the exterior boundary surfaces (WallSurface, RoofSurface, GroundSurface, OuterFloorSurface and OuterCeilingSurface) (Figure 4-35), the ClosureSurface, and the BridgeInstallation. Consequently, these semantic features are the LoD2 child resources of the "brid" resource. The URIs with regard to boundary surfaces are "walls", "roofs", "grounds", "floors" and "ceilings" respectively and retrieve a list of the corresponding thematic surfaces (see Annex A.1). Moreover, with regard to the BridgeInstallation, the "installation" child resource is defined. This resource can be filtered using several filters such as usage, function, class and type (see Annex A.2). Finally, the "closures" resource is embedded so that the open sides of bridge can be virtually closed by using the ClosureSurface. It should be noted that all of the above-mentioned sub-resources have as child resource the respective gmlid value and hence any specific semantic feature can be requested.

```
../brid/{gmlid}/{boundary surfaces}/{gmlid}
```

```
../brid/{gmlid}/{installation}/{gmlid}
```

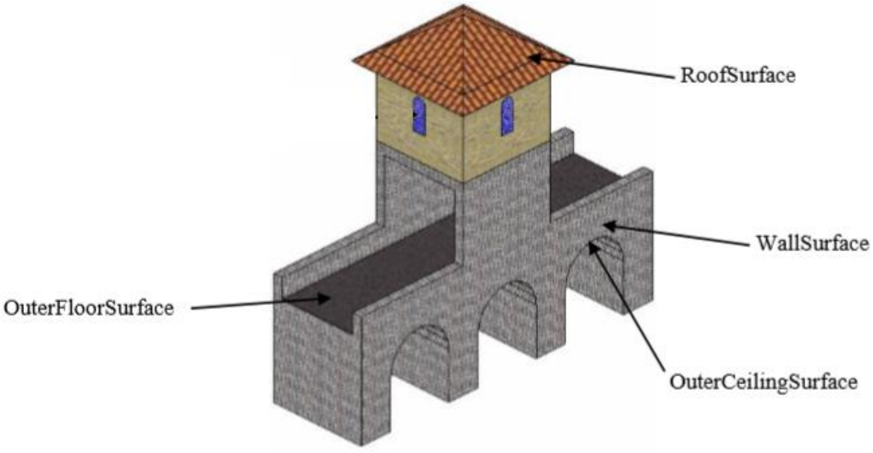


Figure 4-35: Boundary surfaces of a bridge

The conceptual design of "brid" resource with regard to LoD2 is schematically shown in Figure 4-36

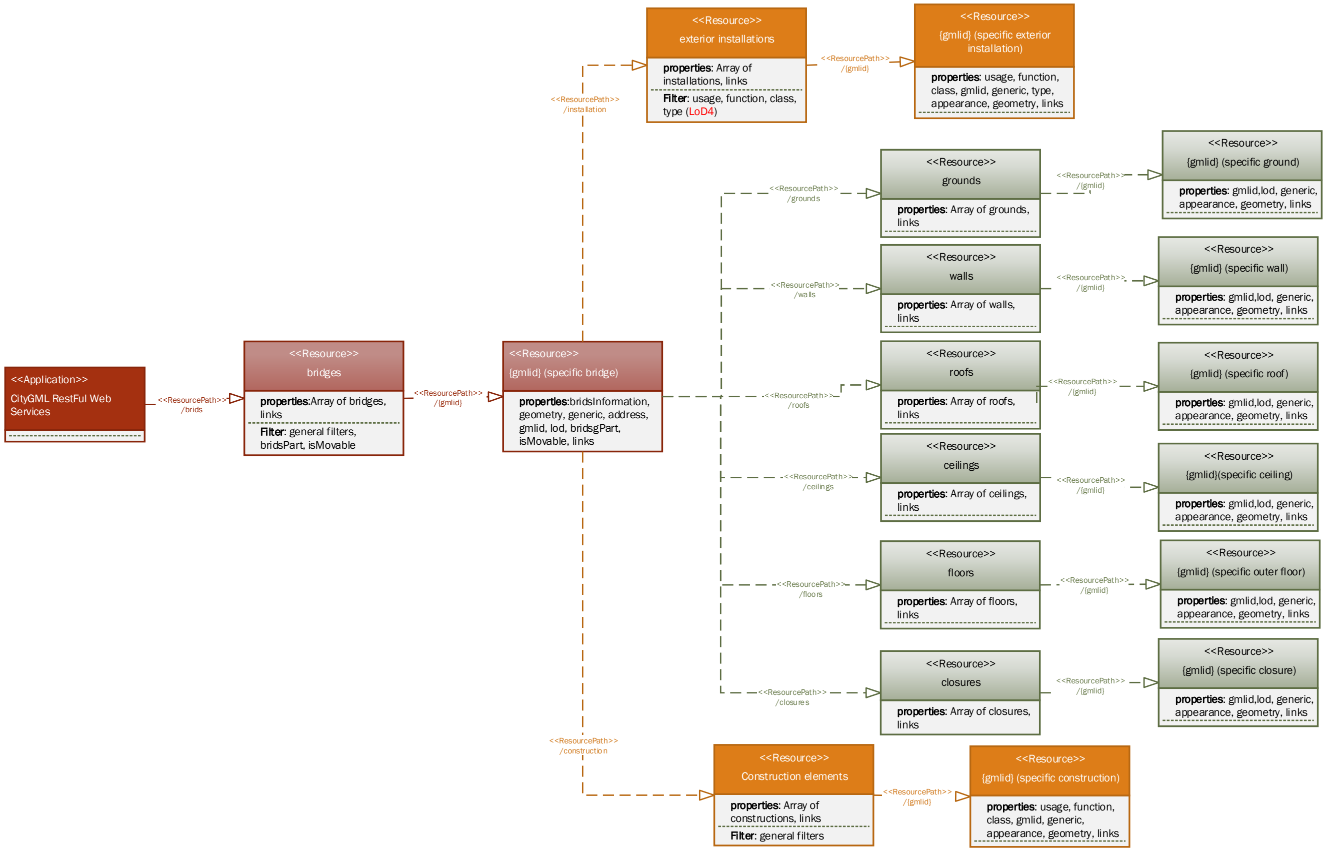

<<Application>>
CityGML RestFul Web Services
<<ResourcePath>> /brids
<<Resource>>
bridges
properties:Array of bridges, links
Filter: general filters, bridsPart, isMovable
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific bridge)
properties:bridsInformation, geometry, generic, address, gmlid, lod, bridsgPart, isMovable, links
<<ResourcePath>> /installation
<<Resource>>
exterior installations
properties: Array of installations, links
Filter: usage, function, class, type (LoD4)
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific exterior installation)
properties: usage, function, class, gmlid, generic, type, appearance, geometry, links
<<ResourcePath>> /grounds
<<Resource>>
grounds
properties: Array of grounds, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific ground)
properties: gmlid,lod, generic, appearance, geometry, links
<<ResourcePath>> /walls
<<Resource>>
walls
properties: Array of walls, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific wall)
properties: gmlid,lod, generic, appearance, geometry, links
<<ResourcePath>> /roofs
<<Resource>>
roofs
properties: Array of roofs, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific roof)
properties: gmlid,lod, generic, appearance, geometry, links
<<ResourcePath>> /ceilings
<<Resource>>
ceilings
properties: Array of ceilings, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid}(specific ceiling)
properties: gmlid,lod, generic, appearance, geometry, links
<<ResourcePath>> /floors
<<Resource>>
floors
properties: Array of floors, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific outer floor)
properties: gmlid,lod, generic, appearance, geometry, links
<<ResourcePath>> /closures
<<Resource>>
closures
properties: Array of closures, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific closure)
properties: gmlid,lod, generic, appearance, geometry, links
<<ResourcePath>> /construction
<<Resource>>
Construction elements
properties: Array of constructions, links
Filter: general filters
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific construction)
properties: usage, function, class, gmlid, generic, appearance, geometry, links

Figure 4-36: Conceptual design of the LoD2 "brid" sub-resource

CHAPTER 4: LOD-BASED THEMATIC RESOURCES

4.3.3. LoD3 brid sub-resources

The additional semantic features of the LoD3 bridge module are the opening features such as windows and doors and hence, the respective resources of the aforesaid features are "windows" and "doors". These resources are child resources of each LoD3 "walls" and "roofs" sub-resources with regard to "brid" main resource (see Annex A.3). The retrieval of specific data regarding the aforementioned resources is achieved implementing the corresponding gmlid as an endpoint. Additionally, each specific "door" resource should contain information regarding the address and, therefore, this object is defined with allowable values in compliance with xAL specification. The conceptual design of the above-mentioned "brid" sub-resources regarding the LoD3 is presented in Figure 4-37.

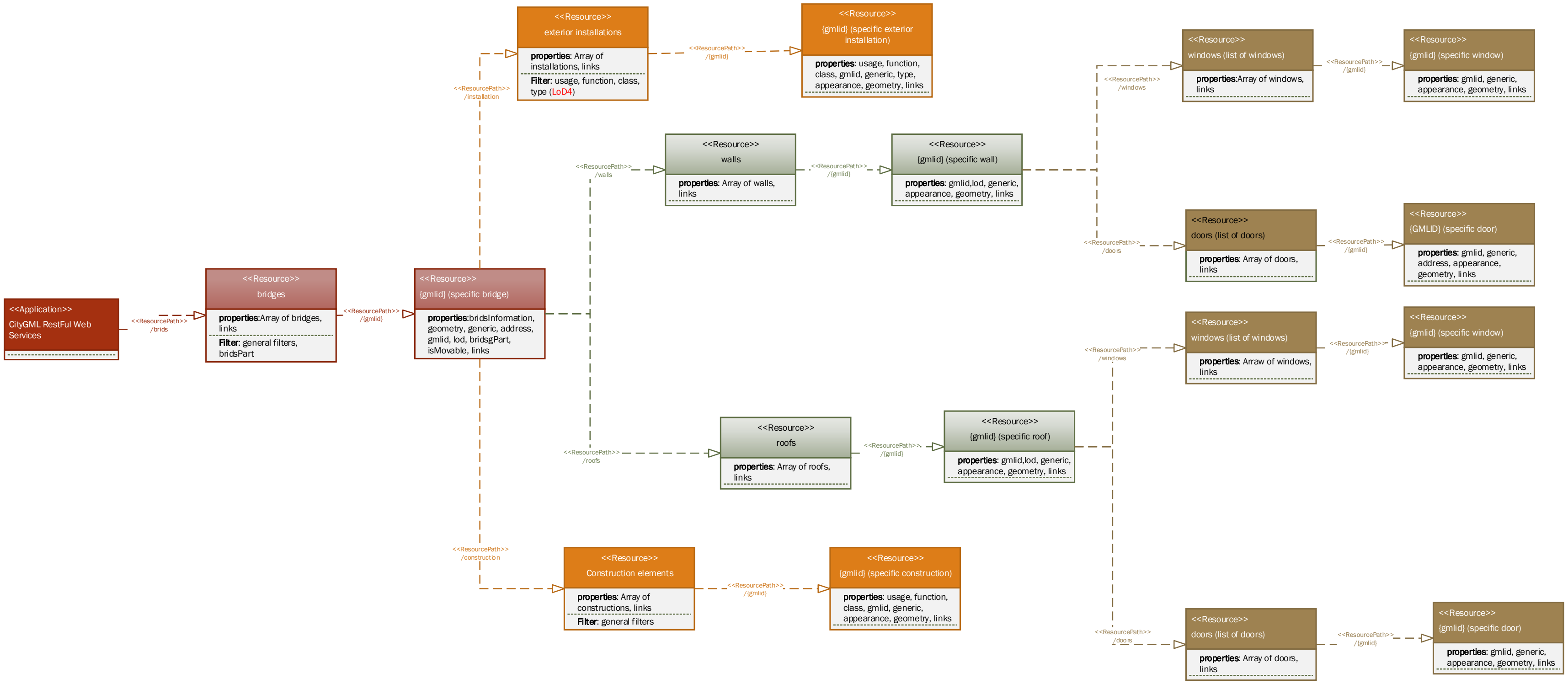

<<Application>>
CityGML RestFul Web Services
<<ResourcePath>>
/brids
<<Resource>>
bridges
properties:Array of bridges, links
Filter: general filters, bridsPart
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific bridge)
properties:bridsInformation, geometry, generic, address, gmlid, lod, bridsgPart, isMovable, links
<<ResourcePath>>
/installation
<<Resource>>
exterior installations
properties: Array of installations, links
Filter: usage, function, class, type (LoD4)
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific exterior installation)
properties: usage, function, class, gmlid, generic, type, appearance, geometry, links
<<ResourcePath>>
/walls
<<Resource>>
walls
properties: Array of walls, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific wall)
properties: gmlid,lod, generic, appearance, geometry, links
<<ResourcePath>>
/windows
<<Resource>>
windows (list of windows)
properties:Array of windows, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific window)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>>
/doors
<<Resource>>
doors (list of doors)
properties: Array of doors, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{GMLID} (specific door)
properties: gmlid, generic, address, appearance, geometry, links
<<ResourcePath>>
/windows
<<Resource>>
windows (list of windows)
properties: Arraw of windows, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific window)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>>
/roofs
<<Resource>>
roofs
properties: Array of roofs, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific roof)
properties: gmlid,lod, generic, appearance, geometry, links
<<ResourcePath>>
/construction
<<Resource>>
Construction elements
properties: Array of constructions, links
Filter: general filters
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific construction)
properties: usage, function, class, gmlid, generic, appearance, geometry, links
<<ResourcePath>>
/doors
<<Resource>>
doors (list of doors)
properties: Array of doors, links
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid} (specific door)
properties: gmlid, generic, appearance, geometry, links

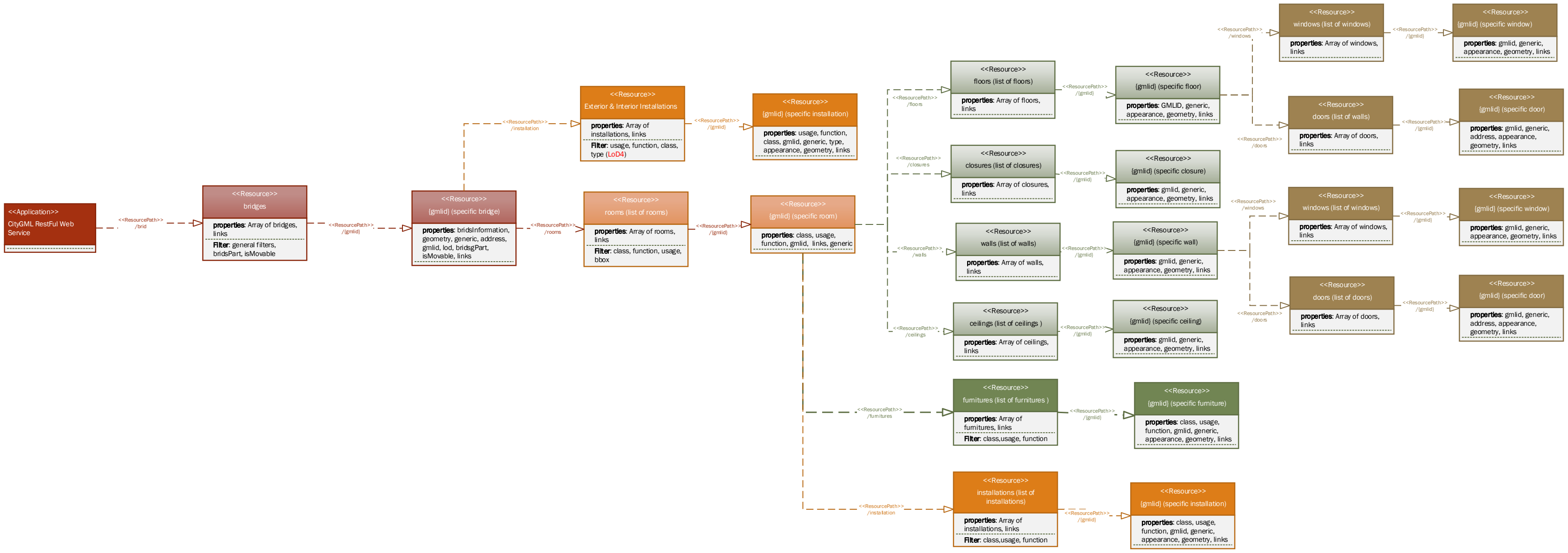
Figure 4-37: Conceptual design of the LoD3 "brid" sub-resources

4.3.4. LoD4 brid sub-resources

In LoD4, the property "type" of the sub-resource "installation" is enabled so that the separation of the interior and exterior installations is achieved. Moreover, the "rooms" child resource is defined and the respective list of the available rooms of a bridge can be retrieved (see Annex A.4). Thereafter, each room provides several links for child resources such as "furniture", "installation", "closures" and boundary surfaces ("walls", "floors" and "ceilings"). The first one retrieves a list of furniture that are located in specific room. The accessible information of this resource is class, usage, function, gmlid, generic, appearance, geometry and links (see Annex A.5). Additionally, the available filter parameters of this resource are class, usage and function. In this context, the rest child resources such as "installation", "walls", "floors" and "ceilings" retrieve a list of the respective available semantic features (see Annex A.2 for "installation" resource; see Annex A.1 for boundary surface resources). Moreover, the "closures" child resource is also embedded so that the opening space that is not filled by a door or window can be sealed by a virtual surface called ClosureSurface. Generally, the retrieval of a particular feature is achieved using the respective gmlid sub-resource. Furthermore, in LoD4, there are two sub-resources with same name but different endpoints. The name of these resources is called "installation". The first one is child resource of "brid" main resource and retrieve a list of interior installation in a specific bridge, while the second one is the child resource of the "rooms" resource and retrieve the respective installations that are located in a specific room (see Annex A.2 for both "installation" sub-resources).

Similar to the LoD3, the interior boundary resources such as "walls" and "floors" provide the "windows" and "doors" child resources which have similar properties and filters like LoD3 opening resources (see Annex A.3).

The conceptual design of the additional sub-resources of the LoD4 "brid" resource is shown in Figure 4-38



<<Application>>
CityGML RestFul Web Service
<<ResourcePath>> /brid
<<Resource>>
bridges
properties: Array of bridges, links
Filter: general filters, bridsPart, isMovable
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific bridge)
properties: bridsInformation, geometry, generic, address, gmlid, lod, bridsgPart, isMovable, links
<<ResourcePath>> /installation
<<Resource>>
Exterior & Interior Installations
properties: Array of installations, links
Filter: usage, function, class, type (LoD4)
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific installation)
properties: usage, function, class, gmlid, generic, type, appearance, geometry, links
<<ResourcePath>> /rooms
<<Resource>>
rooms (list of rooms)
properties: Array of rooms, links
Filter: class, function, usage, bbox
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific room)
properties: class, usage, function, gmlid, links, generic
<<ResourcePath>> /floors
<<Resource>>
floors (list of floors)
properties: Array of floors, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific floor)
properties: GMLID, generic, appearance, geometry, links
<<ResourcePath>> /closures
<<Resource>>
closures (list of closures)
properties: Array of closures, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific closure)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>> /walls
<<Resource>>
walls (list of walls)
properties: Array of walls, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific wall)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>> /ceilings
<<Resource>>
ceilings (list of ceilings)
properties: Array of ceilings, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific ceiling)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>> /windows
<<Resource>>
windows (list of windows)
properties: Array of windows, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific window)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>> /doors
<<Resource>>
doors (list of walls)
properties: Array of doors, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific door)
properties: gmlid, generic, address, appearance, geometry, links
<<ResourcePath>> /windows
<<Resource>>
windows (list of windows)
properties: Array of windows, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific window)
properties: gmlid, generic, appearance, geometry, links
<<ResourcePath>> /doors
<<Resource>>
doors (list of doors)
properties: Array of doors, links
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific door)
properties: gmlid, generic, address, appearance, geometry, links
<<ResourcePath>> /furnitures
<<Resource>>
furnitures (list of furnitures)
properties: Array of furnitures, links
Filter: class,usage, function
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific furniture)
properties: class, usage, function, gmlid, generic, appearance, geometry, links
<<ResourcePath>> /installation
<<Resource>>
installations (list of installations)
properties: Array of installations, links
Filter: class,usage, function
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid} (specific installation)
properties: class, usage, function, gmlid, generic, appearance, geometry, links

Figure 4-38: Conceptual design of the LoD4 "brid" sub-resources

4.3.5. Case studies using semantic requests

This section presents the use of the resources with respect to the bridge model of CityGML v2. Therefore, a CityGML dataset is utilized which contains a variety of different types of bridges in different LoDs (Figure 4-39)

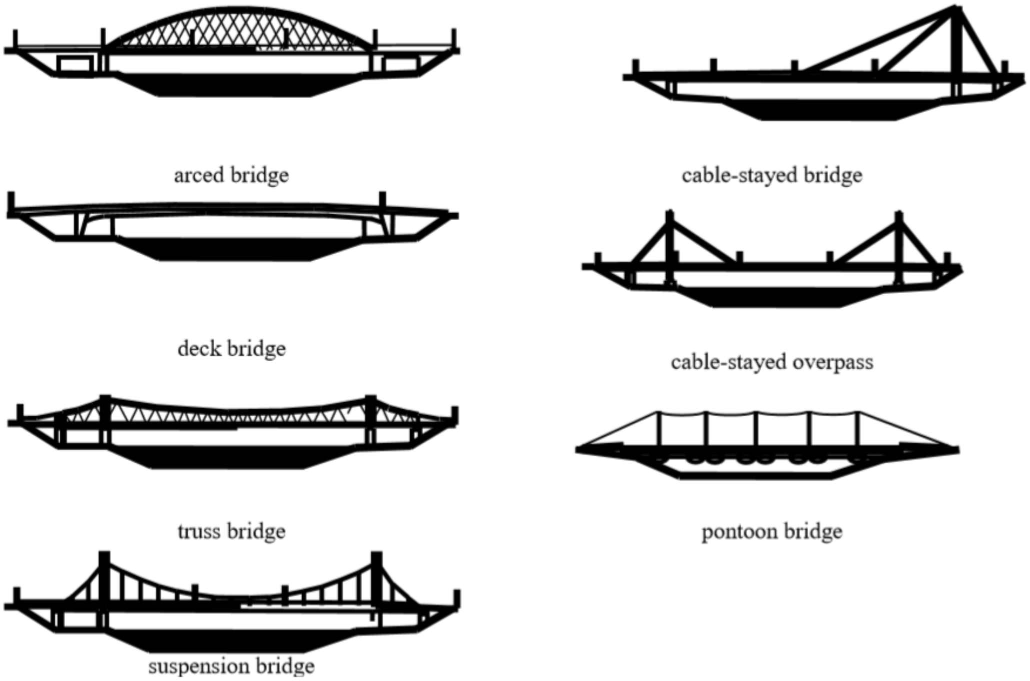
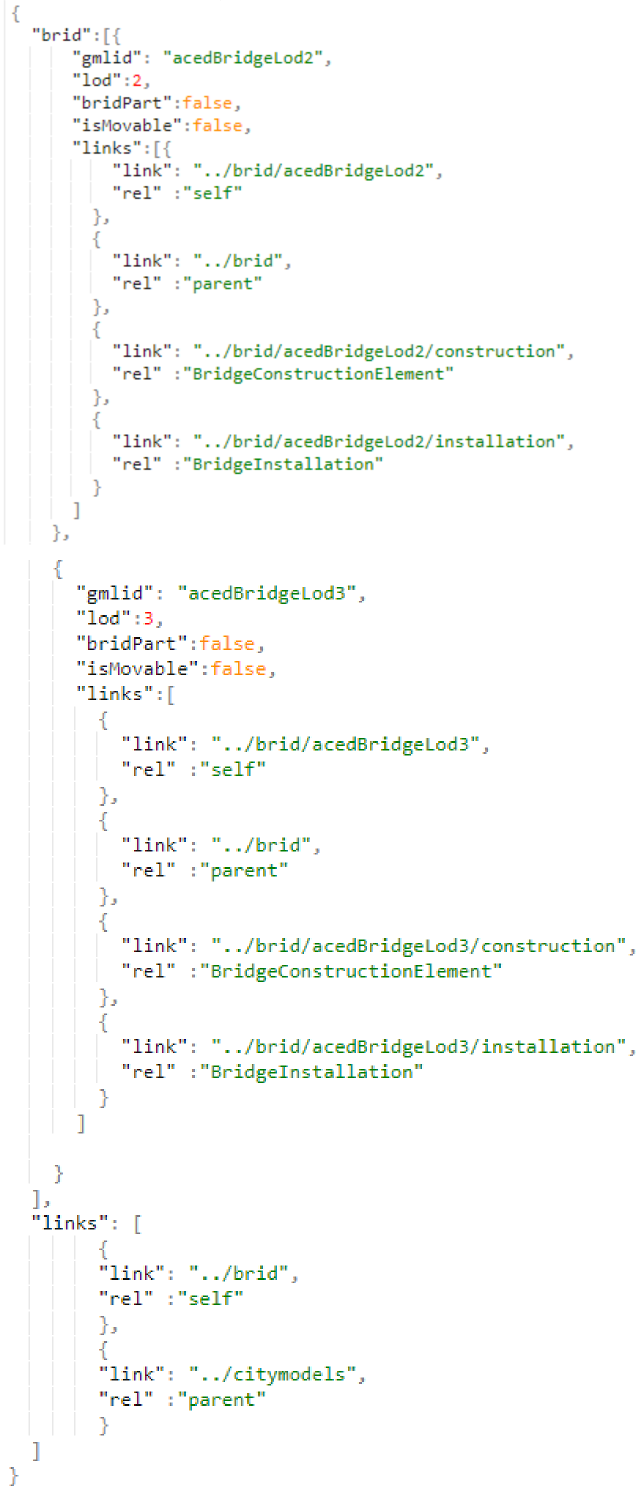


Figure 4-39: Different types of bridges

(Gröger et al., 2012)

The above categories are based on the available code list of the class attribute of the CityGML bridge module (Gröger et al., 2012). So, implementing the first request, the available arced bridges (class: 1000) are retrieved (Figure 4-40).

```
../brid?class=1000
```

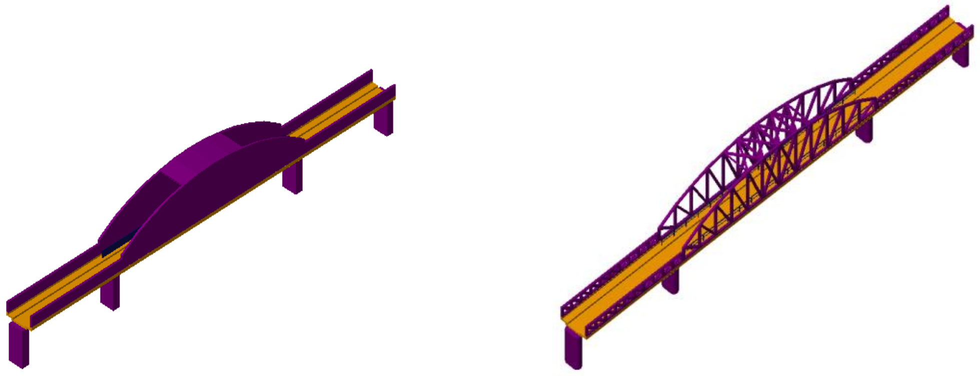


```
{
  "brid":[{
      "gmlid": "acedBridgeLod2",
      "lod":2,
      "bridPart":false,
      "isMovable":false,
      "links":[{
          "link": "../brid/acedBridgeLod2",
          "rel" :"self"
        },
        {
          "link": "../brid",
          "rel" :"parent"
        },
        {
          "link": "../brid/acedBridgeLod2/construction",
          "rel" :"BridgeConstructionElement"
        },
        {
          "link": "../brid/acedBridgeLod2/installation",
          "rel" :"BridgeInstallation"
        }
      ]
    },
    {
      "gmlid": "acedBridgeLod3",
      "lod":3,
      "bridPart":false,
      "isMovable":false,
      "links":[
        {
          "link": "../brid/acedBridgeLod3",
          "rel" :"self"
        },
        {
          "link": "../brid",
          "rel" :"parent"
        },
        {
          "link": "../brid/acedBridgeLod3/construction",
          "rel" :"BridgeConstructionElement"
        },
        {
          "link": "../brid/acedBridgeLod3/installation",
          "rel" :"BridgeInstallation"
        }
      ]

    }
  ],
  "links": [
        {
        "link": "../brid",
        "rel" :"self"
        },
        {
        "link": "../citymodels",
        "rel" :"parent"
        }
  ]
}
```

Figure 4-40: JSON-based result for aced bridges

According to the said result (see Figure 4-40), there are two aced bridges available in LoD2 and LoD3 respectively which are not bridge parts and are also immovable (Figure 4-41).



(a) LoD2 aced bridge

(b) LoD3 aced bridge

Figure 4-41: Results of same aced bridge in different LoDs: (a) LoD2, (b) LoD3

Moreover, for more information on each bridge the corresponding endpoints can be requested:

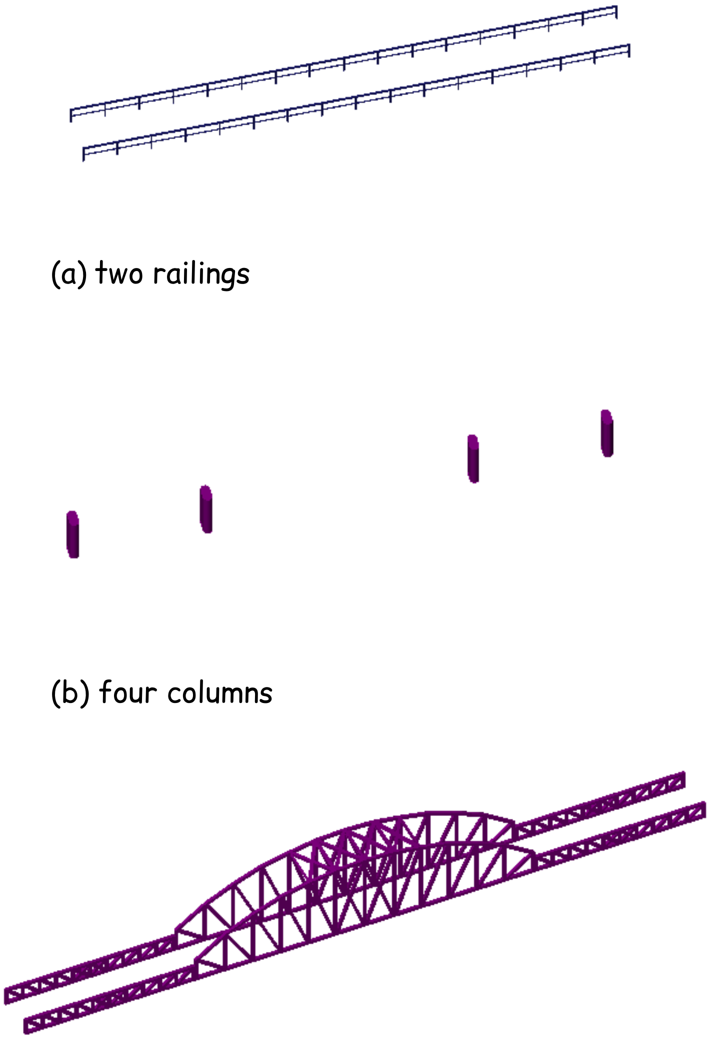
```
../brid/acedBridgeLod2                                    ../brid/ acedBridgeLod3
```

Furthermore, for both bridges, there are only two sub-resources such as "installation" and "construction". Thereafter, focusing on retrieving information about the installations and constructions of LoD3 aced bridges the following URIs are requested:

```
../brid/acedBridgeLod3/installation
```

```
../brid/ acedBridgeLod3/construction
```

Totally, there are two installations available (two railings) (Figure 4-42-(a)) and eleven construction elements (four columns and seven additional constructions) (Figure 4-42-(b) & (c)).



(a) two railings

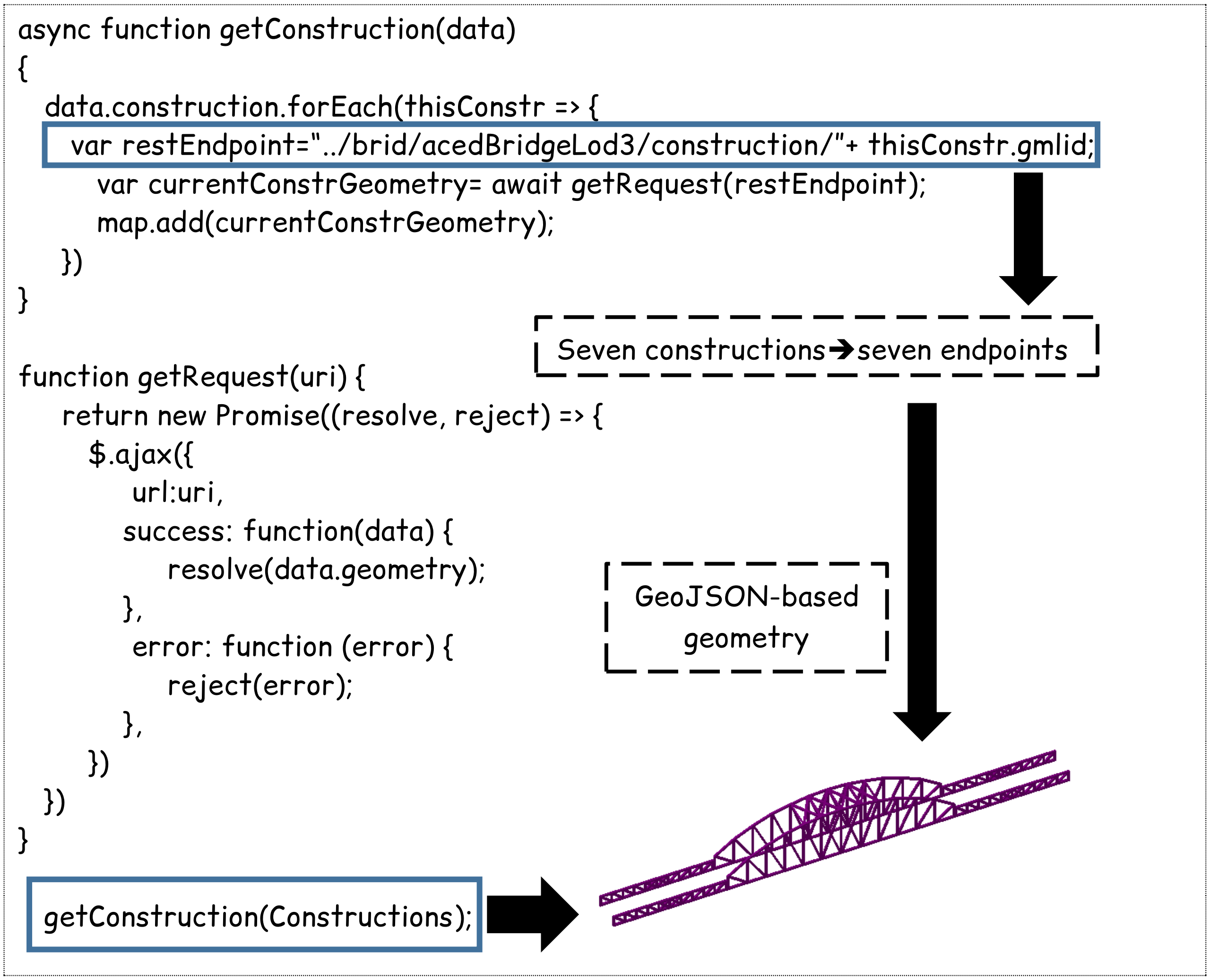
(b) four columns

(c) seven additional construction elements

Semantic themes	gmlid
Railing	instRail1
Railing	instRail2
collumn	instCol1
collumn	instCol2
collumn	instCol3
collumn	instCol4
Construction	instConst1
Construction	instConst2
Construction	instConst3
Construction	instConst4
Construction	instConst5
Construction	instConst6
Construction	instConst7

Figure 4-42: Result of construction and installation semantic elements of the LoD3 bridge instance

Finally, seven additional endpoints should be requested to fetch and visualize all the available construction elements (see Figure 4-42-(c)) using the respective gmlid of each element as sub-resource. For this purpose, the following JavaScript code is implemented (Figure 4-43)



```
async function getConstruction(data)
{
  data.construction.forEach(thisConstr => {
    var restEndpoint="../brid/acedBridgeLod3/construction/"+ thisConstr.gmlid;
      var currentConstrGeometry= await getRequest(restEndpoint);
      map.add(currentConstrGeometry);
   })
}

function getRequest(uri) {
    return new Promise((resolve, reject) => {
      $.ajax({
          url:uri,
         success: function(data) {
             resolve(data.geometry);
         },
          error: function (error) {
             reject(error);
         },
      })
  })
}

getConstruction(Constructions);
```

Figure 4-43: JavaScript-based procedure to request all the construction elements of a specific LoD3 aced bridge

5. LOD-INDEPENDENT THEMATIC RESOURCES

This chapter presents the conceptual design of the rest of the main resources of the CityGML RESTful Web service which are mainly LoD-independent thematic resources. Namely, these resources are enriched with semantic characteristics either independently of LoD or from LoD2 and above without any different from one level to another. Hence, the thematic resources with same availability in all LoD as well as the thematic resources with same availability from LoD2 and above are presented.

In this chapter, the 4th sub-research question of the current dissertation is partially answered:

> *How could CityGML data be semantically retrieved by users without knowledge of the source?*

This chapter is based on the following paper:

> **Pispidikis** and Dimopoulou (2019)

CHAPTER 5: LOD-INDEPENDENT THEMATIC RESOURCES

5.1. Thematic Resources Available in all LoDs

The thematic modules of CityGML v2 allow the representation of the thematic and spatial parameters of the 3D models' objects at different levels of detail. The transition from one level to another imposes and allows different semantic details both on the outside and inside. Consequently, the sub-resources for the main resources are designed based on LoD. However, the majority of the thematic modules of CityGML v2 are enriched with semantic characteristics either independently of LoD or from LoD2 and above without any different from one level to another. Hence, the sub-resources of these main resources will be available for all LoDs or from LoD2 and above.

The main resources that their semantic features are independent of LoD are the "grp", "dem", "frn", "luse", "veg", "vegetation" and "plantcovers". These resources are conceptual designed according to the corresponding thematic modules of CityGML v2 such as "CityObjectGroup", "Relief", "CityFurniture", "LandUse", "Vegetation" and the additional sub-classes of the "Vegetation" module like "SolitaryVegetationObject" and "PlantCover".

5.1.1. Veg resources

The Vegetation features are important components of a 3D city model, since they support the recognition of the surrounding environment. These objects of CityGML v2 distinguish between solitary vegetation object like trees and vegetation areas, which represent biotopes like forest or other plant communities (Figure 5-1). These features can be requested using the "veg" main resource which provides information about the available vegetation objects grouped on the basis of the aforementioned categories such as "vegetation" (solitary vegetation objects) and "plantcovers" (vegetation areas). Also, this resource can be filtered using a new filter parameter called "vegetationtype with values according to the aforementioned categories (Figure 5-2).

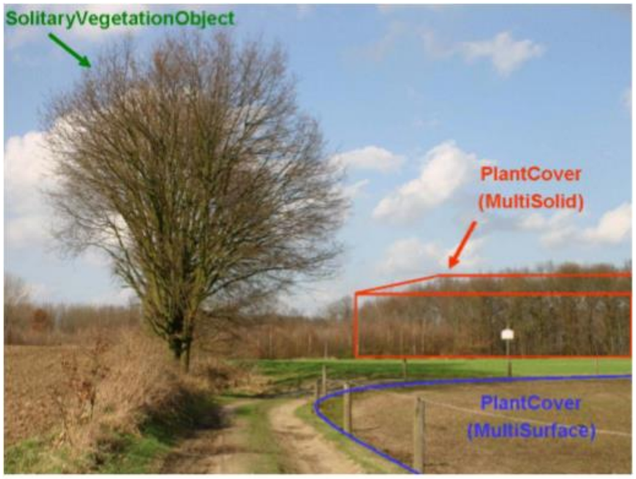
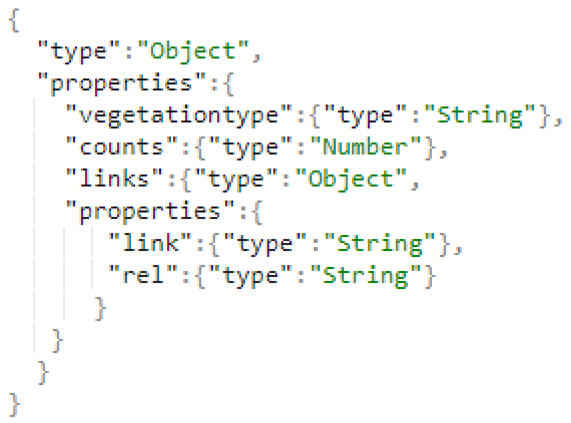


Figure 5-1: Example for vegetation objects of the sub-classes SolitaryVegetationObject and PlantCover

(Gröger et al., 2012)

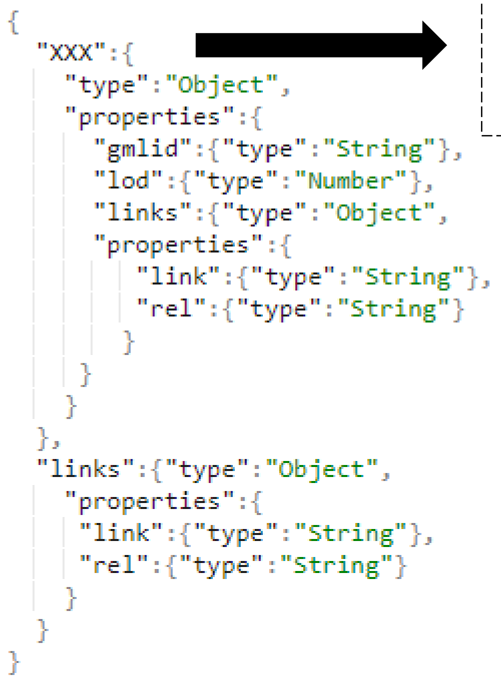


```
{
  "type":"Object",
  "properties":{
    "vegetationtype":{"type":"String"},
    "counts":{"type":"Number"},
    "links":{"type":"Object",
    "properties":{
       "link":{"type":"String"},
       "rel":{"type":"String"}
      }
    }
  }
}
```

Figure 5-2: JSON-based schema of "veg" resource

Additionally, in each group, the corresponding resource links of the available vegetation models are provided. As a result, the "veg" main resource is mainly used in order to inform the users about the two available group resources regarding the solitary vegetation objects and the vegetation areas. So, the URI resources of these categories are "vegetation" and plantcovers" respectively. These resources are not sub-resources, since they are independent resources of the "veg" resource. Therefore, the JSON-based schema of these resources is presented in Figure 5-3

../{vegetation or plantcovers}



```
{
  "XXX":{
    "type":"Object",
    "properties":{
      "gmlid":{"type":"String"},
      "lod":{"type":"Number"},
      "links":{"type":"Object",
      "properties":{
         "link":{"type":"String"},
         "rel":{"type":"String"}
        }
      }
    }
  },
  "links":{"type":"Object",
    "properties":{
     "link":{"type":"String"},
     "rel":{"type":"String"}
    }
  }
}
```

The character "XXX" has the value of either vegetation or plantcover. Also, the retrieval data is array of features that belongs to the corresponding category.

Figure 5-3: JSON-based schema of "vegetation" and "plantcovers" resources

Thereafter, when a particular vegetation model is requested by implementing the respective gmlid, various information is retrieved such as vegInformation, generic, gmlid, lod, links, appearance and geometry. The vegInformation object contains a variety of attributes depending on the category to which the particular vegetation model belongs (Table 5-1).

vegInformation	Type	category
class	Number	vegetation & plantcovers
usage	Number	vegetation & plantcovers
function	Number	vegetation & plantcovers
species	Number	vegetation
height	Number	vegetation
trunkDiameter	Number	vegetation
crownDiameter	Number	vegetation

averageHeight	Number	plantcovers

Table 5-1: Available attributes of vegInformation object

Furthermore, the general filters (see Table 3-4) can be used in both categories, while the vegetation category can also be filtered using the "species" parameter. The value of this parameter is defined according to the codelist of CityGML specification regarding the solitaryVegetationObject attribute "species".

The conceptual design of the "veg", "vegetation" and "plantcovers" resources with available properties and filters is shown in Figure 5-4.

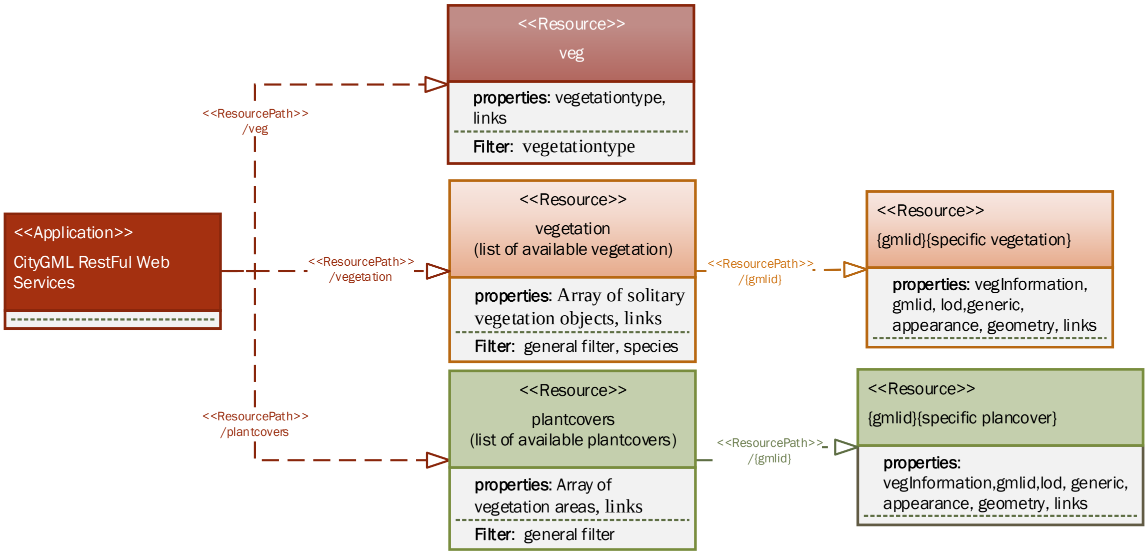
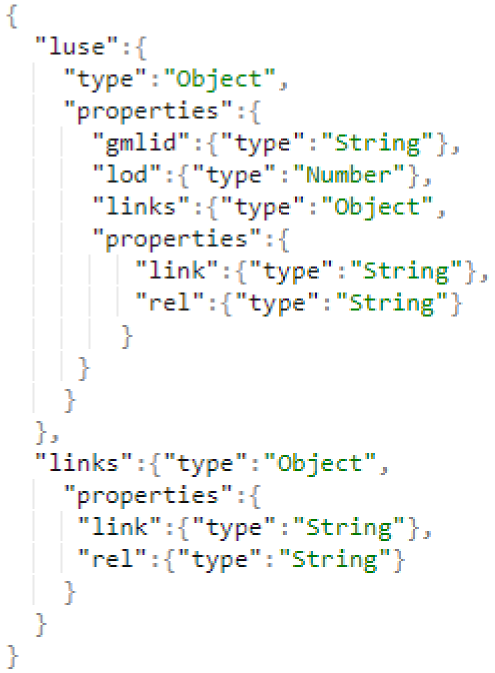

<<Application>>
CityGML RestFul Web Services
<<ResourcePath>>
/veg
<<Resource>>
veg
properties: vegetationtype, links
Filter: vegetationtype
<<ResourcePath>>
/vegetation
<<Resource>>
vegetation
(list of available vegetation)
properties: Array of solitary vegetation objects, links
Filter: general filter, species
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid}{specific vegetation}
properties: vegInformation, gmlid, lod,generic, appearance, geometry, links
<<ResourcePath>>
/plantcovers
<<Resource>>
plantcovers
(list of available plantcovers)
properties: Array of vegetation areas, links
Filter: general filter
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid}{specific plancover}
properties: vegInformation,gmlid,lod, generic, appearance, geometry, links

Figure 5-4: Conceptual design of "veg", "vegetation" and "plantcovers" resources

5.1.2. Luse resources

The "luse" resource retrieves information with regard to the LandUse model of the CityGML v2. This model can be used to describe areas of the earth's surface dedicated to a specific land use, but also to describe areas of the earth's surface having a specific land cover with or without vegetation, such as sand, rock, mud flat, forest, etc. Furthermore, it represents both the land use and the land cover concepts. The first describes the human activities on the earth's surface whereas the second one describes its physical and biological cover. Hence, the retrieval information of the "luse" resource is a list of the aforementioned concepts and also a "links" object which contains links to itself and to parent resources (Figure 5-5)



```
{
  "luse":{
    "type":"Object",
    "properties":{
      "gmlid":{"type":"String"},
      "lod":{"type":"Number"},
      "links":{"type":"Object",
      "properties":{
         "link":{"type":"String"},
         "rel":{"type":"String"}
        }
      }
    }
  },
  "links":{"type":"Object",
    "properties":{
     "link":{"type":"String"},
     "rel":{"type":"String"}
    }
  }
}
```

Figure 5-5: JSON-based schema of "luse" resource

The implementation of a specific "luse" resource provides various information such as luseInformation, lod, gmlid, links, generic, appearance and geometry. Moreover, the general filters (see Table 3-4) are also available.

It should be noted that the LandUse module of CityGML v2 is defined for all LoDs (LoD 0-4) and may have different geometries in any LoD. However, it has no extra semantic characteristics on transition from one LoD to another and thus, except for the gmlid, the "tun" resource is simple URI with no extra sub-resources.

The conceptual design of the “luse” resource is shown in Figure 5-6.

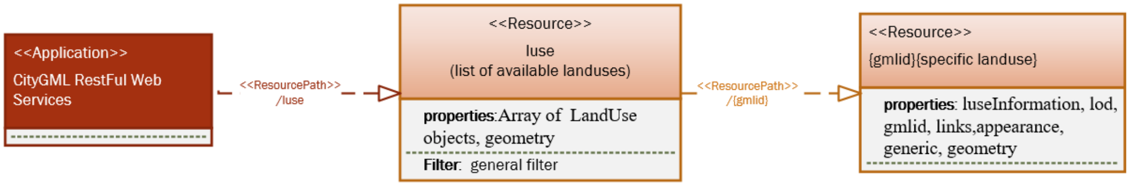

<<Application>>
CityGML RestFul Web Services
<<ResourcePath>>
/luse
<<Resource>>
luse
(list of available landuses)
properties:Array of LandUse objects, geometry
Filter: general filter
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid}{specific landuse}
properties: luseInformation, lod, gmlid, links,appearance, generic, geometry

Figure 5-6: Conceptual design of “luse” resource

5.1.3. Frn resources

The “frn” main resource refers to the city furniture module of the CityGML v2. The objects of this module are immovable objects like lanterns, traffic lights, traffic signs, or bus stops and can be found in traffic areas, residential areas, on squares, or in built-up areas (Figure 5-7).

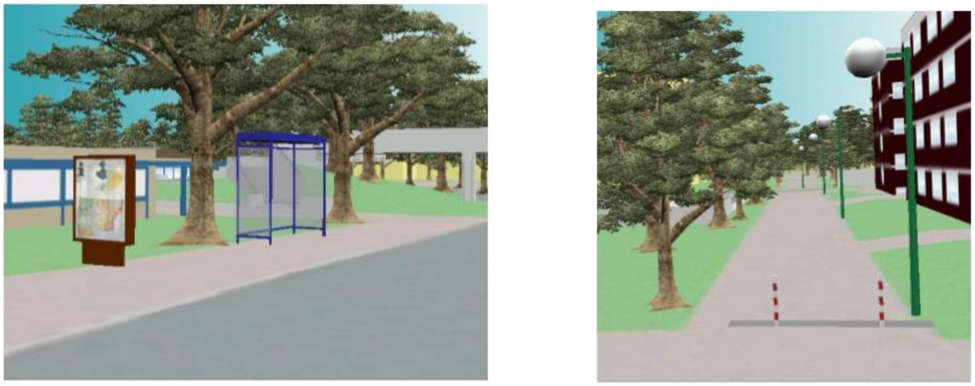


Figure 5-7: City furniture objects

The city furniture objects can be represented in city models with their specific geometry (GeoJSON format), but in most cases the same kind of object has an identical geometry. This means that the geometry of the prototype city furniture is stored only once in a local CRS in all LoDs and referenced by other city furniture features. Hence, in these cases, the implicit object is implemented.

The "frn" resource has similar schema as the "luse" resource and also when a particular city furniture model is requested, various information is retrieved such as frnInformation, lod, gmlid, links, generic, appearance and geometry. The frnInformation contains the attributes class, function and usage. More specific, the class attribute allows object classification like traffic light, traffic sign, delimitation stake or garbage can. Additionally, the function attribute describes, to which thematic area the city furniture feature belongs to e.g. transportation, traffic regulation etc. and the attribute usage denoted the real purpose of the object. Also, the general filters (see Table 3-4) can be utilized.

The conceptual design of the "frn" resource is presented in Figure 5-8

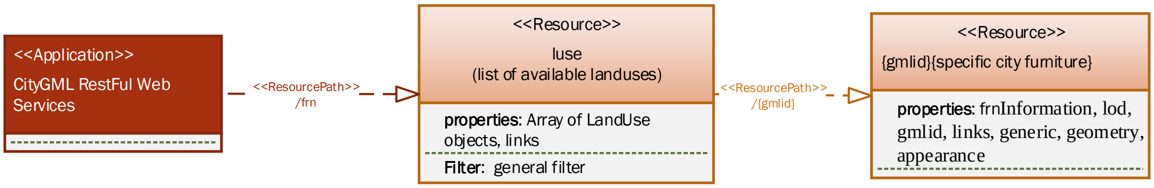
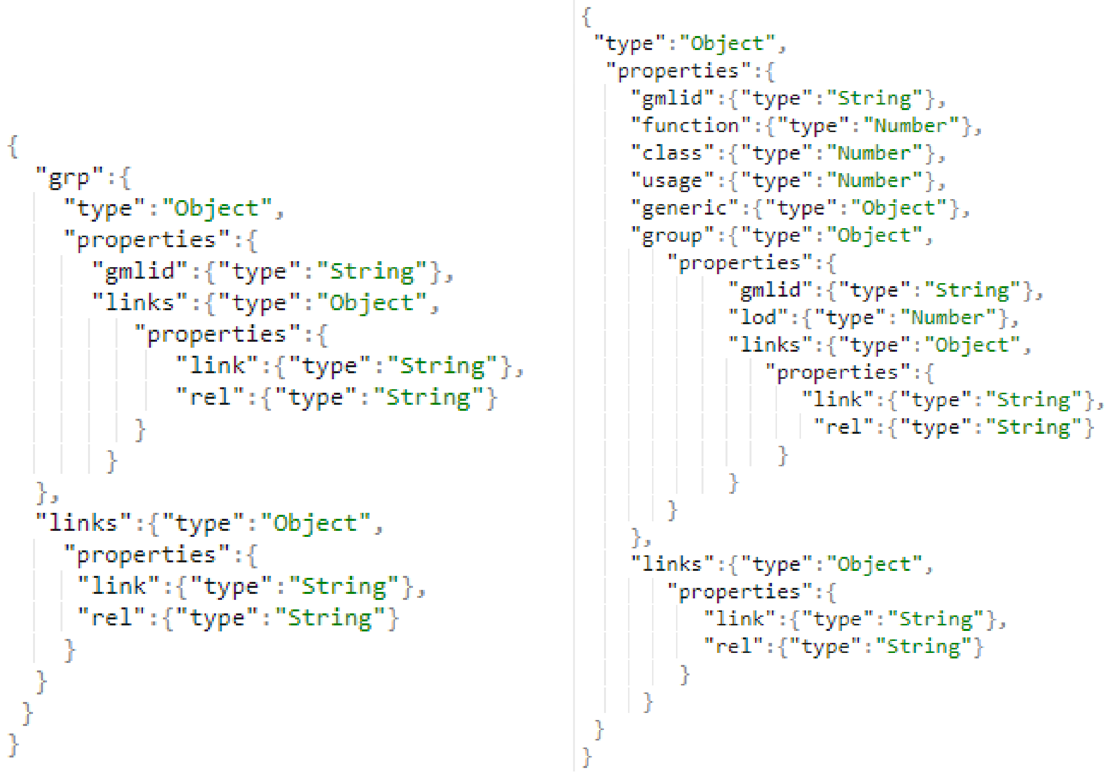

<<Application>>
CityGML RestFul Web Services
<<ResourcePath>>
/frn
<<Resource>>
luse
(list of available landuses)
properties: Array of LandUse objects, links
Filter: general filter
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid}{specific city furniture}
properties: frnInformation, lod, gmlid, links, generic, geometry, appearance

Figure 5-8: Conceptual design of "frn" resource

5.1.4. Grp resources

The CityObjectGroup module delivers the grouping concept of CityGML that allows for the aggregation of arbitrary city objects according to user-defined criteria, and to represent and transfer these aggregations as part of a 3D city model. The endpoint for this resource is the "grp" main resource and is used to retrieve all the available city object groups of a datasets (Figure 5-9(a)). Next, when a particular city object group is requested, a wide range of information is retrieved such as gmlid, generic, function, usage, class and group. The group object contains a list of the grouped main resources of CityGML RESTful Web service simultaneously with their respective links (Figure 5-9(b)).



```
{
  "grp":{
    "type":"Object",
    "properties":{
      "gmlid":{"type":"String"},
      "links":{"type":"Object",
        "properties":{
          "link":{"type":"String"},
          "rel":{"type":"String"}
        }
      }
  },
  "links":{"type":"Object",
    "properties":{
     "link":{"type":"String"},
     "rel":{"type":"String"}
    }
  }
 }
}
```

(a) ../grp

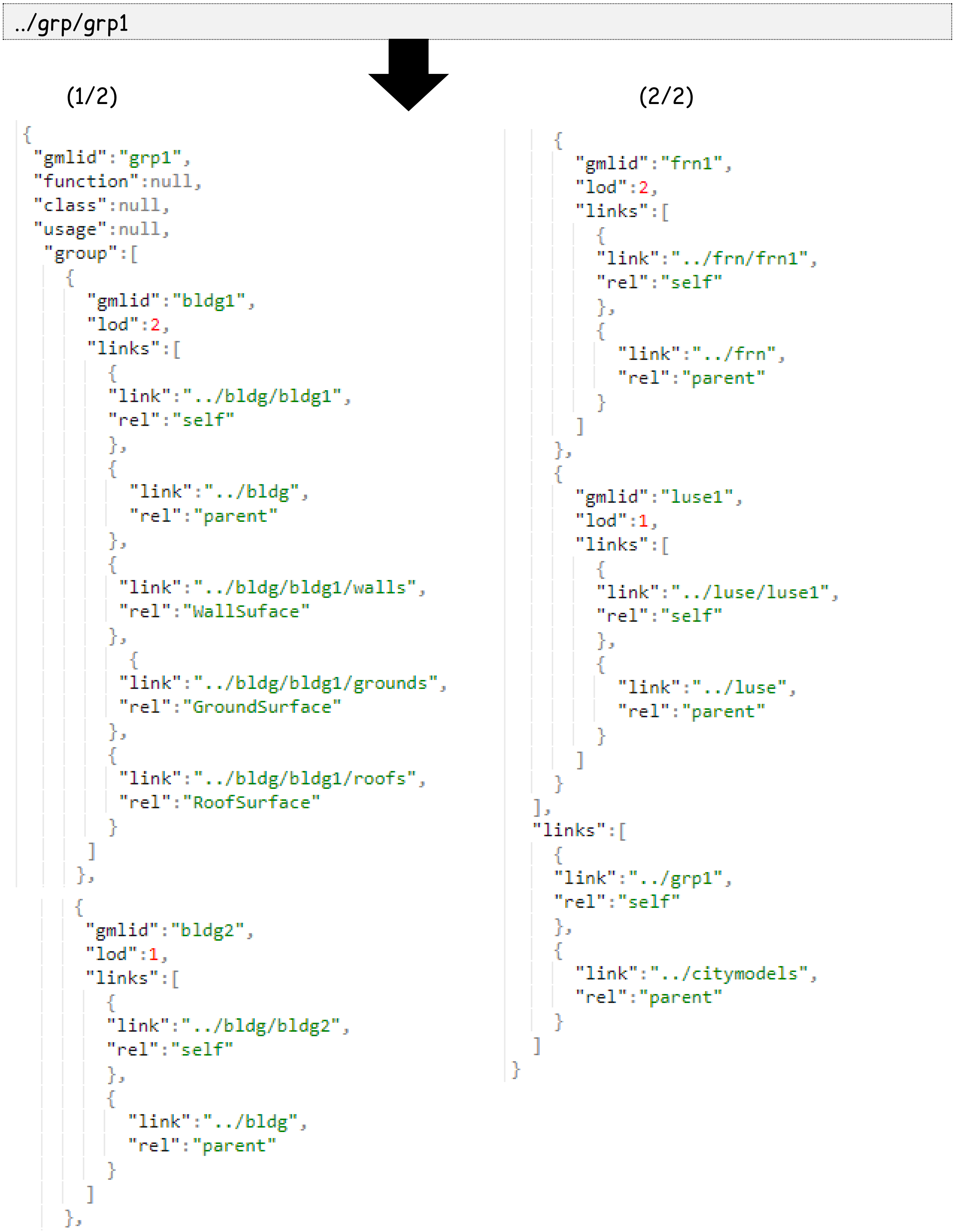
```
{
 "type":"Object",
  "properties":{
    "gmlid":{"type":"String"},
    "function":{"type":"Number"},
    "class":{"type":"Number"},
    "usage":{"type":"Number"},
    "generic":{"type":"Object"},
    "group":{"type":"Object",
      "properties":{
        "gmlid":{"type":"String"},
        "lod":{"type":"Number"},
        "links":{"type":"Object",
          "properties":{
            "link":{"type":"String"},
             "rel":{"type":"String"}
          }
        }
      }
    },
    "links":{"type":"Object",
      "properties":{
        "link":{"type":"String"},
        "rel":{"type":"String"}
      }
    }
  }
}
```

(b)../grp/{gmlid}

Figure 5-9: JSON-based schema of "grp" resource

5.1.4.1. Case study using semantic requests

A CityGML dataset contains a city object group (gmlid: grp1), which groups a variety of objects such as two buildings (gmlid:bldg1, lod:2 & gmlid:bldg2, lod: 1), one city furniture (gmlid: frn1, lod: 2) and one LandUse (gmlid: luse1, lod: 1). The information about the available objects of this city object group can be retrieved by implementing the specific "grp" main resource as follows (Figure 5-10). Additionally, the corresponding endpoints of these objects are also provided to be utilized for further requests.



../grp/grp1

(1/2)

```
{
 "gmlid":"grp1",
 "function":null,
 "class":null,
 "usage":null,
  "group":[
    {
      "gmlid":"bldg1",
      "lod":2,
      "links":[
        {
        "link":"../bldg/bldg1",
        "rel":"self"
        },
        {
          "link":"../bldg",
          "rel":"parent"
        },
        {
         "link":"../bldg/bldg1/walls",
         "rel":"WallSuface"
        },
          {
         "link":"../bldg/bldg1/grounds",
         "rel":"GroundSurface"
        },
        {
         "link":"../bldg/bldg1/roofs",
         "rel":"RoofSurface"
        }
      ]
    },
     {
      "gmlid":"bldg2",
      "lod":1,
      "links":[
        {
        "link":"../bldg/bldg2",
        "rel":"self"
        },
        {
          "link":"../bldg",
          "rel":"parent"
        }
      ]
    },
```

(2/2)

```
    {
      "gmlid":"frn1",
      "lod":2,
      "links":[
        {
        "link":"../frn/frn1",
        "rel":"self"
        },
        {
          "link":"../frn",
          "rel":"parent"
        }
      ]
    },
    {
      "gmlid":"luse1",
      "lod":1,
      "links":[
        {
        "link":"../luse/luse1",
        "rel":"self"
        },
        {
          "link":"../luse",
          "rel":"parent"
        }
      ]
    }
  ],
  "links":[
    {
    "link":"../grp1",
    "rel":"self"
    },
    {
      "link":"../citymodels",
      "rel":"parent"
    }
  ]
}
```

Figure 5-10: JSON-based result for city object group instance

The conceptual design of the "grp" resource is presented in Figure 5-11

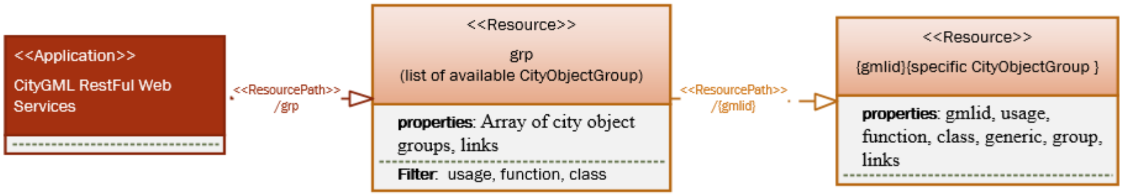
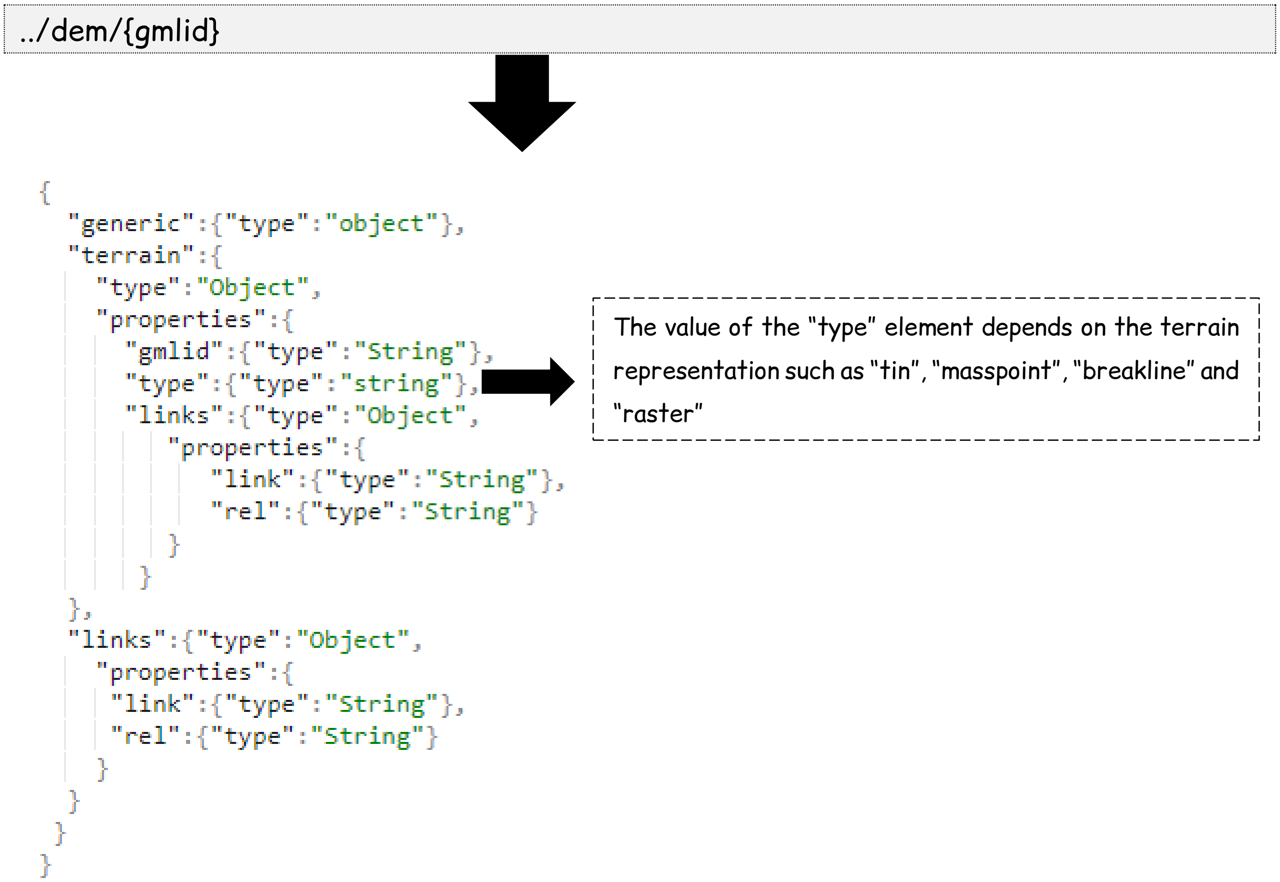

<<Application>>
CityGML RestFul Web Services
<<ResourcePath>>
/grp
<<Resource>>
grp
(list of available CityObjectGroup)
properties: Array of city object groups, links
Filter: usage, function, class
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid}{specific CityObjectGroup }
properties: gmlid, usage, function, class, generic, group, links

Figure 5-11: Conceptual design of "grp" resource

5.1.5. Dem resources

An essential part of a city model is the terrain. Therefore, the Digital Terrain Model (DTM) of CityGML v2 is provided by the thematic module "Relief". Additionally, in CityGML, the terrain is associated with different concepts of terrain representations which can coexist. Specifically, the terrain may be specified as a regular raster or grid, as a Triangulated Irregular Network (TIN), by break lines, or by a mass points. These four terrain types may be combined in different ways, yielding a high flexibility. Firstly, each type may be represented in different LoDs, reflecting different accuracies or resolutions. Moreover, a terrain can be described by the combination of multiple types, for example by a raster and break lines, or by a TIN and break lines etc.

The information about "Relief" module of CityGML can be requested by implementing the "dem" main resource. In section 5.1, the main resources that their semantic features are independent of LoD are presented. However, apart from the "dem" resource, the rest of the said main resources have no extra sub-resources except for gmlid. Consequently, the "dem" resource is used to retrieve a list of available reliefs and, thereafter, when a specific relief is requested, then four sub-resources are available such as "tins", "masspoints", "breaklines" and "raster". Hence, the implementation of a specific relief retrieves the "links", the "generic" objects and also a list of objects according to the available sub-resources. (Figure 5-12).



../dem/{gmlid}

```
{
  "generic":{"type":"object"},
  "terrain":{
    "type":"Object",
    "properties":{
      "gmlid":{"type":"String"},
      "type":{"type":"string"},
      "links":{"type":"Object",
         "properties":{
            "link":{"type":"String"},
            "rel":{"type":"String"}
         }
       }
  },
  "links":{"type":"Object",
    "properties":{
     "link":{"type":"String"},
     "rel":{"type":"String"}
    }
  }
 }
}
```

The value of the "type" element depends on the terrain representation such as "tin", "masspoint", "breakline" and "raster"

Figure 5-12: JSON-based schema of "dem" resource

Also, the above- mentioned retrieval result can be filtered using various filters such as lod, bbox and type. These filters are mainly used to limit the elements of the "terrain" object.

Furthermore, the geometry object of a "tins" sub-resource could be a set of either triangles or control points, break and stop lines. Moreover, the geometry object of "breaklines" sub-resource can be composed of break lines and ridge/valley lines. The break lines indicate abrupt changes of terrain slope, while the ridge/value lines in addition mark a change of the sign of the terrain slope gradient.

The conceptual design of the "dem" resource and its sub-resources is schematically shown in Figure 5-13

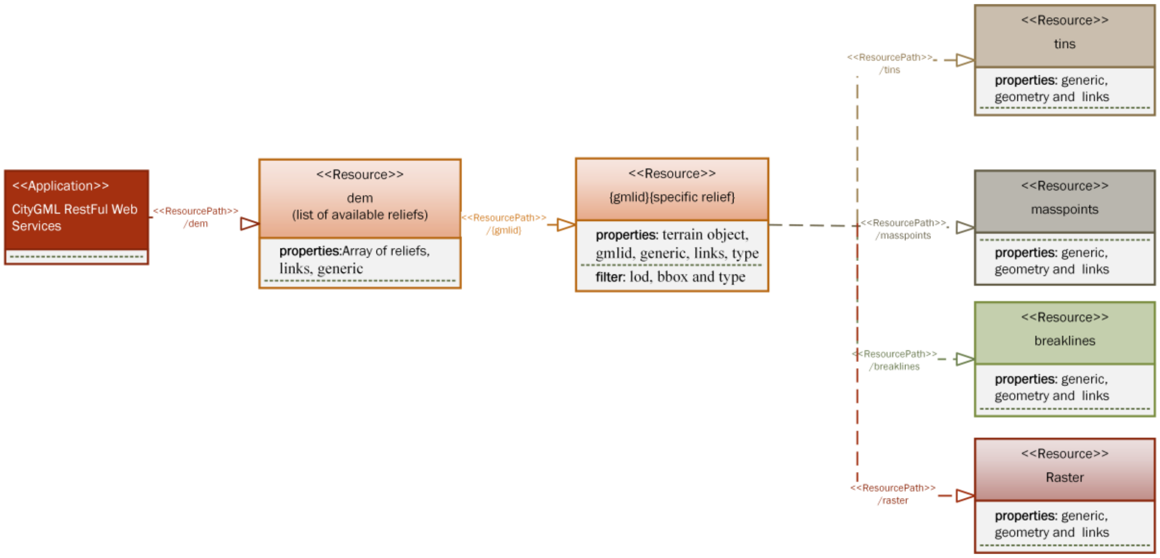

<<Application>>
CityGML RestFul Web Services
<<ResourcePath>>
/dem
<<Resource>>
dem
(list of available reliefs)
properties:Array of reliefs, links, generic
<<ResourcePath>>
/{gmlid}
<<Resource>>
{gmlid}{specific relief}
properties: terrain object, gmlid, generic, links, type
filter: lod, bbox and type
<<ResourcePath>>
/tins
<<Resource>>
tins
properties: generic, geometry and links
<<ResourcePath>>
/masspoints
<<Resource>>
masspoints
properties: generic, geometry and links
<<ResourcePath>>
/breaklines
<<Resource>>
breaklines
properties: generic, geometry and links
<<ResourcePath>>
/raster
<<Resource>>
Raster
properties: generic, geometry and links

Figure 5-13: Conceptual design of "dem" resource

5.2. Thematic Resources Available from LoD2 and Above

The "Transportation" and "WaterBody" modules of CityGML v2 belong to the category of models that their semantic enrichment is available and same from LoD2 and above. Consequently, the sub-resources of the respective "tun" and "wtr" main resources of CityGML RESTful Web service are only available from LoD2 and above, without any differentiation.

5.2.1. Tran resources

The transportation module of CityGML is a multi-functional and multi-scale model focusing on thematic, functional, geometric and topological aspects of a road. According to CityGML v2, the road is represented as a "TransportationComplex" which has different geometrical representation through the different LoDs.

In LoD0, the transportation complexes are modelled by line objects establishing a linear network. In case of areal transportation objects like squares, they should be modelled in the same way as in Geographic Data Files (GDF) (ISO, 2011), which is used in most car navigation systems. Specifically, in GDF, a square is represented as a ring surrounding the place and to which the incident roads connect (Figure 5-14)

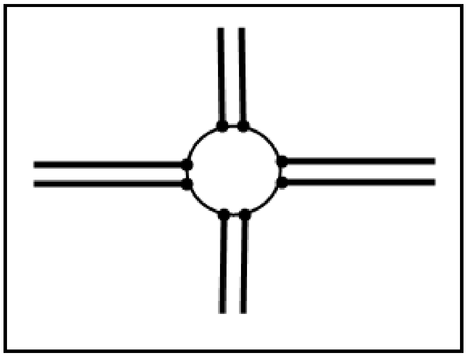


Figure 5-14: Representation of roundabout

In LoD1, all transportation features are geometrically described by 3D surfaces, while in LoD2-LoD4, the transportation complexes are further subdivided thematically into "TrafficAreas" and "AuxiliraryTrafficAreas" (Figure 5-15)

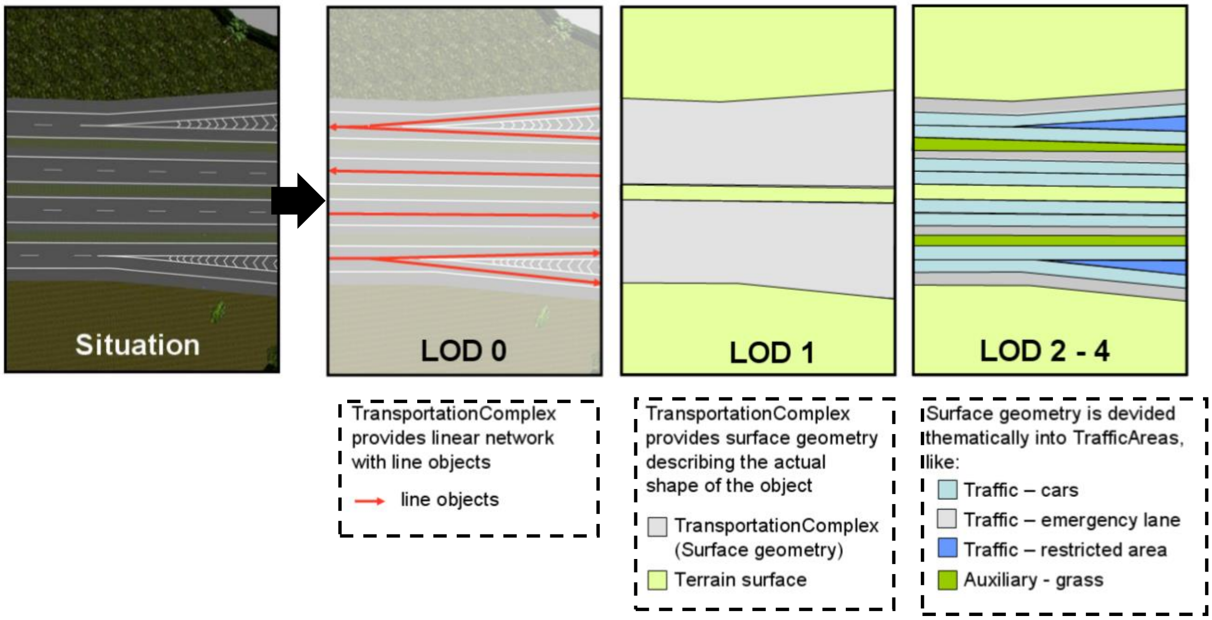
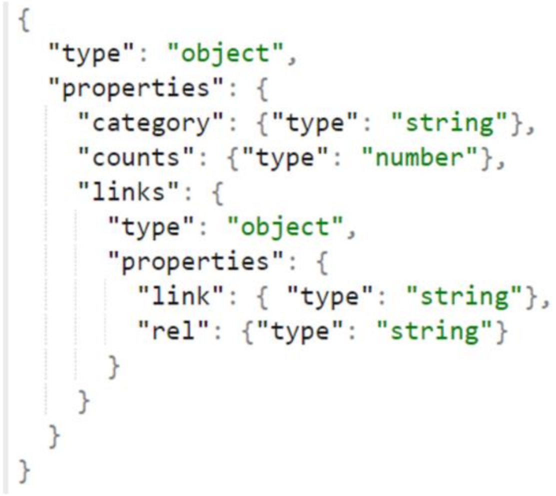

Situation
LOD 0
LOD 1
LOD 2 - 4
TransportationComplex provides linear network with line objects
→ line objects
TransportationComplex provides surface geometry describing the actual shape of the object
□ TransportationComplex (Surface geometry)
□ Terrain surface
Surface geometry is devided thematically into TrafficAreas, like:
□ Traffic – cars
□ Traffic – emergency lane
■ Traffic – restricted area
■ Auxiliary - grass

Figure 5-15: Transportation model representation in different LoDs

The transportation module can be requested by implementing the "tran" main resource of CityGML RESTful Web service. This resource is mainly used to provide information about the four available group resources in accordance with the sub-classes of CityGML transportation module such as road, track, railway and square. Therefore, the information retrieval of this resource is the available transportation models grouped by the said predefined sub-classes (Figure 5-16).



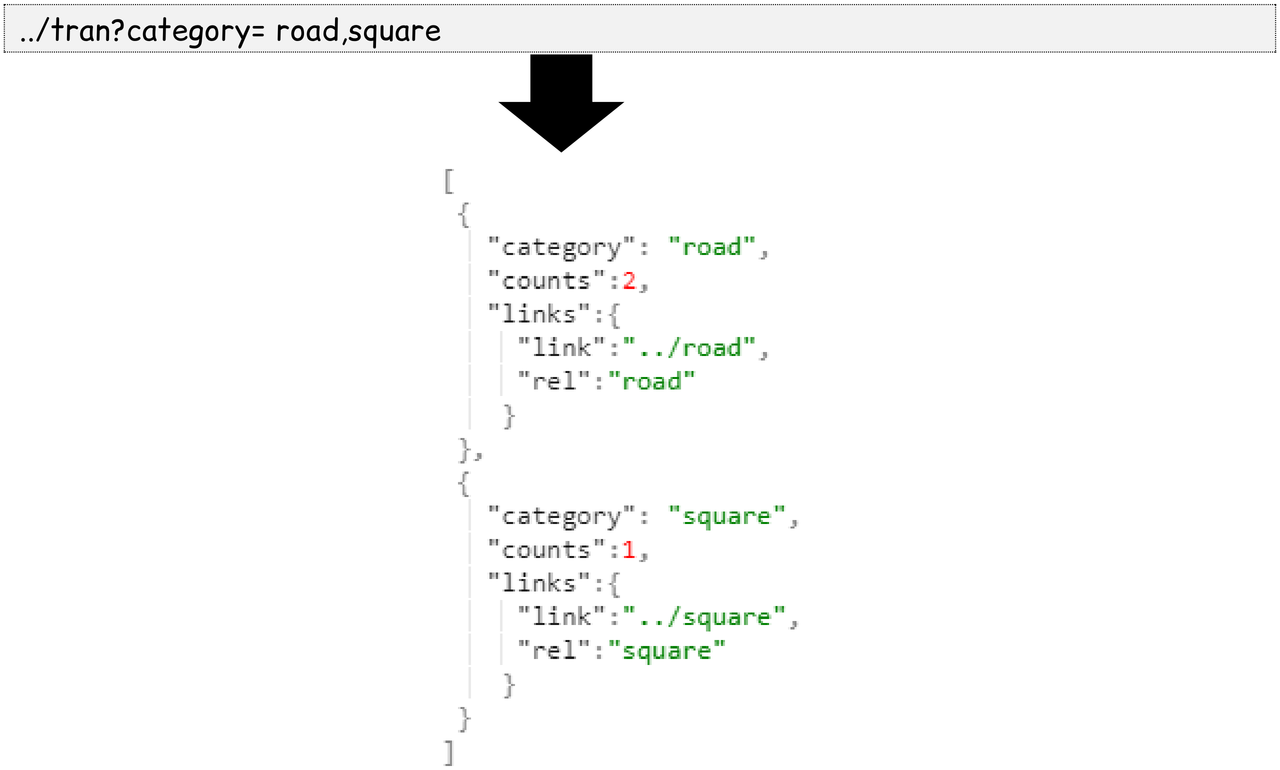
```
{
  "type": "object",
  "properties": {
    "category": {"type": "string"},
    "counts": {"type": "number"},
    "links": {
      "type": "object",
      "properties": {
        "link": { "type": "string"},
        "rel": {"type": "string"}
      }
    }
  }
}
```

Figure 5-16: JSON-based schema of "tran" main resource

Also, the "tran" main resource can be filtered using a filter parameter called "category" with value the respective above-mentioned sub-classes of transportation module. It should be noted that multi-category values can be implemented simultaneously separating them with comma punctuation. Thereafter, in each group category, the corresponding resource link of the specific transportation model can also be retrieved.

```
../tran?category= road,square
```

For instance, if a CityGML dataset contains a variety of transportation complexes in different LoDs such as one track, one railway, one square and one road (LoD0 & LoD1) then the following response is retrieved (Figure 5-17)



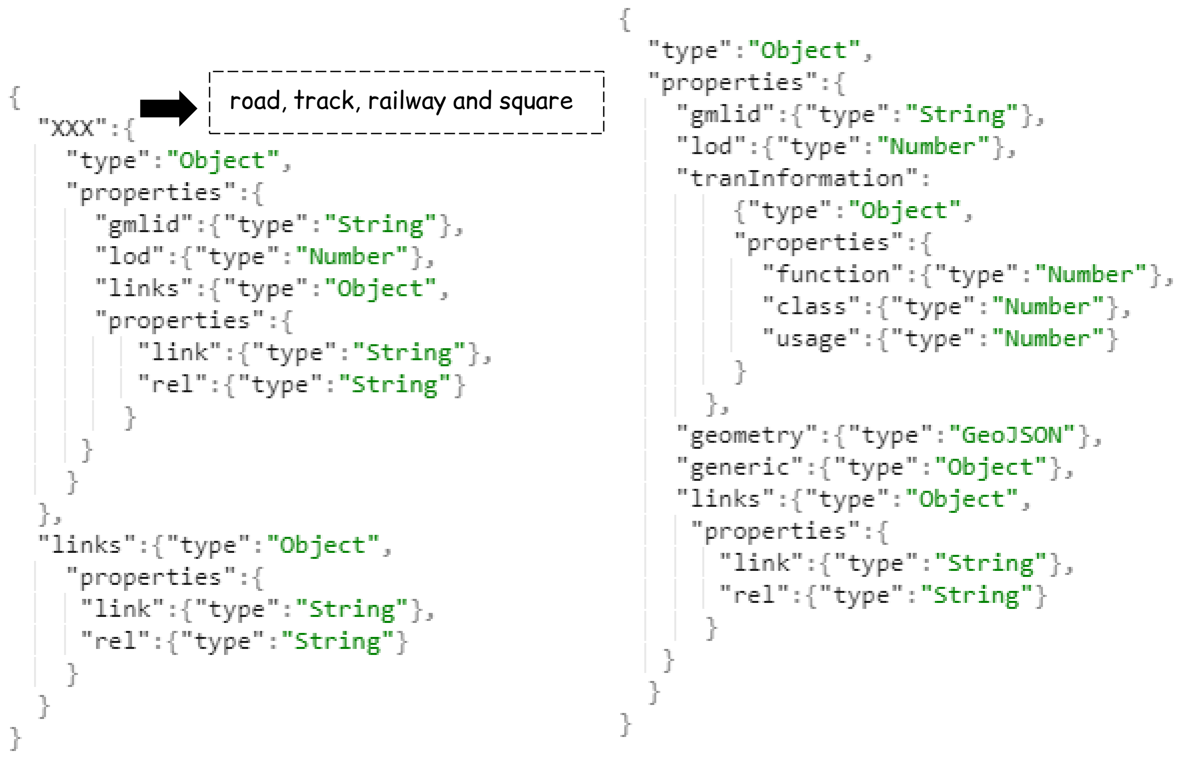
```
../tran?category= road,square
```

```
[
 {
   "category": "road",
   "counts":2,
   "links":{
     "link":"../road",
     "rel":"road"
    }
 },
 {
   "category": "square",
   "counts":1,
   "links":{
     "link":"../square",
     "rel":"square"
    }
 }
]
```

Figure 5-17: JSON result by using “tran” main resource

It should be noted that in the above-mentioned example, although there is one road, there are two instances of this road based on the corresponding LoD.

The four predefined sub-classes are conceptual designed as extra main resources and not as sub-resources, since they are independent of the “tran” main resource (Figure 5-18(a)). Additionally, in LoD0 and LoD1, there are no extra semantic characteristics for these resources and thus, the gmlid is only their child resource. Thereafter, when this child resource is implemented, various properties are retrieved such as tranInformation, generic, gmlid, lod, links and geometry (Figure 5-18(b)).



road, track, railway and square

```
{
  "XXX":{
    "type":"Object",
    "properties":{
      "gmlid":{"type":"String"},
      "lod":{"type":"Number"},
      "links":{"type":"Object",
      "properties":{
         "link":{"type":"String"},
         "rel":{"type":"String"}
        }
      }
    }
  },
  "links":{"type":"Object",
    "properties":{
     "link":{"type":"String"},
     "rel":{"type":"String"}
    }
  }
}
```

```
{
  "type":"Object",
  "properties":{
    "gmlid":{"type":"String"},
    "lod":{"type":"Number"},
    "tranInformation":
        {"type":"Object",
        "properties":{
          "function":{"type":"Number"},
          "class":{"type":"Number"},
          "usage":{"type":"Number"}
        }
      },
    "geometry":{"type":"GeoJSON"},
    "generic":{"type":"Object"},
    "links":{"type":"Object",
     "properties":{
       "link":{"type":"String"},
       "rel":{"type":"String"}
      }
    }
  }
}
```

a) …/{extra main resources}

b) …/{extra main resources}/{gmlid}

Figure 5-18: JSON-based schema of extra main resources (road, track, railway and square)

From LoD2 and above the four available resources (road, track, railway and square) are further subdivided semantically into TrafficAreas, which are used by transportation, such as cars, trains, public transport, airplanes, bicycles or pedestrian and in AuxiliaryTrafficAreas, which are of minor importance for transportation purposes (Figure 5-19). The URIs of these child resources are "trafficareas" and "auxiliaries" respectively and are used to retrieve all their available thematic surfaces (see Annex A.7). Next, when a particular thematic surface is requested, the following information is retrieved: class, usage, function, surfaceMaterial, lod, generic, gmlid and geometry (see Annex A.7). Moreover, the aforementioned list can be filtered by implementing the general filters (see Table 3-4).

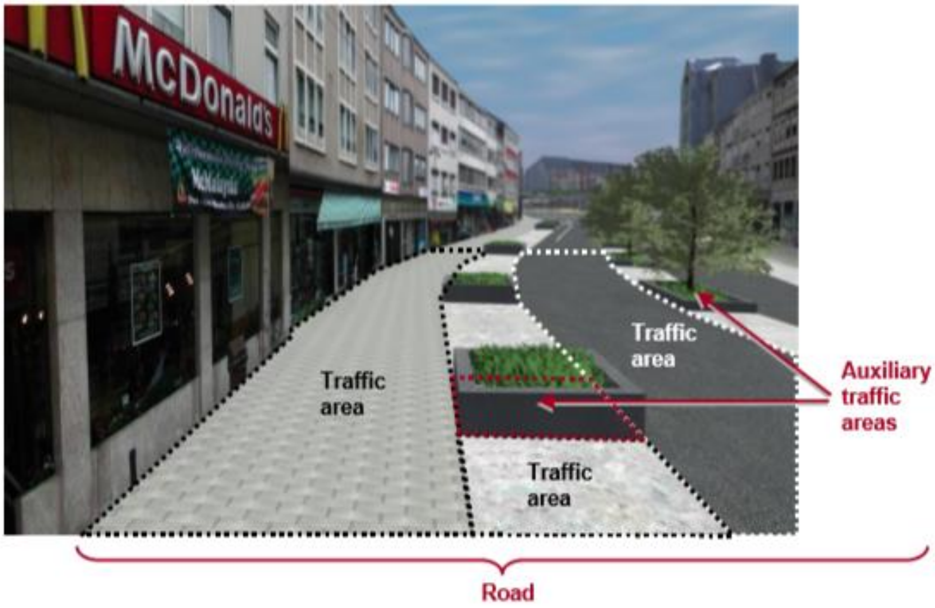


Figure 5-19: Example for the representation of LoD2 transportation module in CityGML using TrafficAreas and AuxiliaryTrafficAreas

(Gröger et al., 2012)

As a result, the conceptual design of the tran, road, square, railway and track main resources with regard to transportation module is presented in Figure 5-20

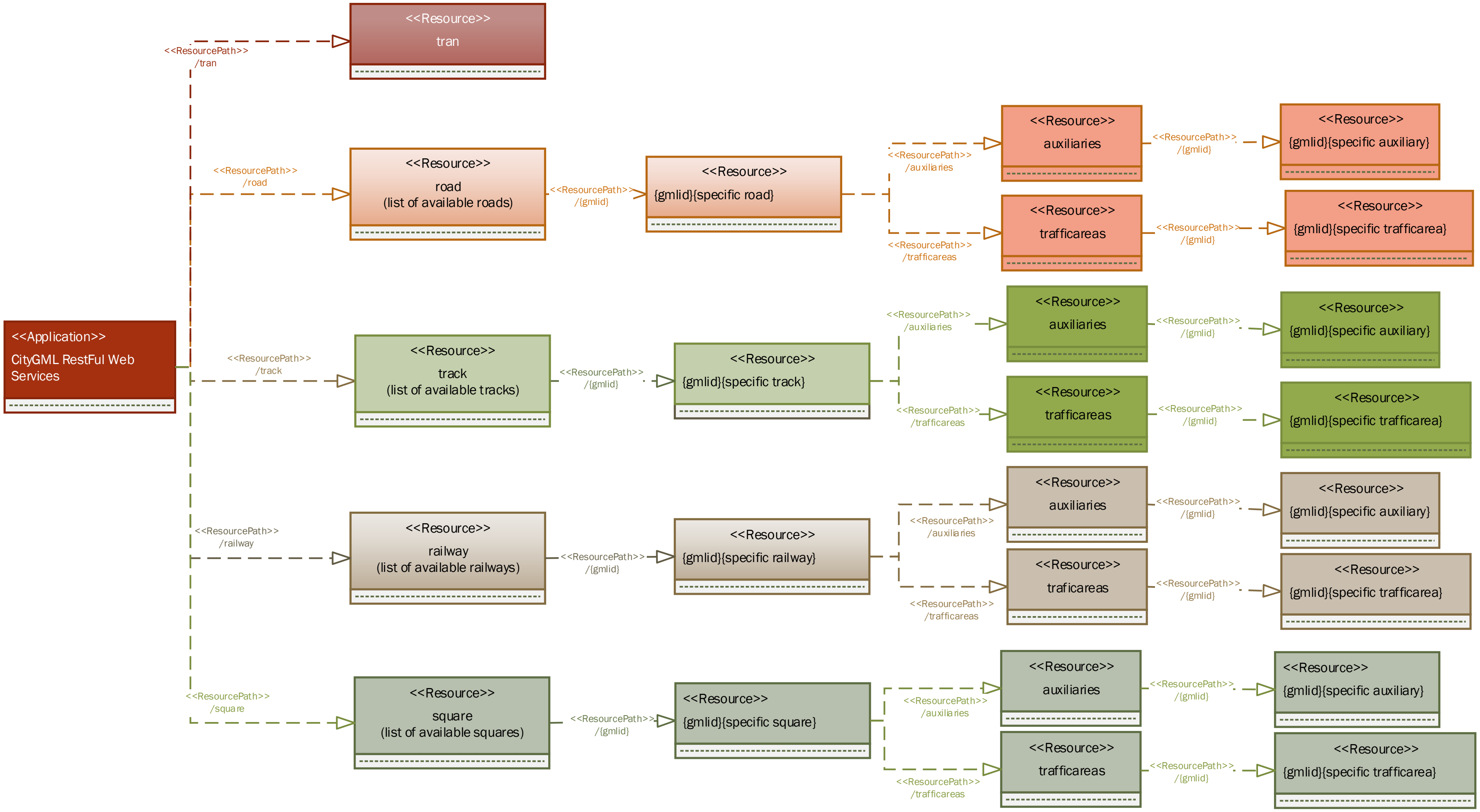


Figure 5-20: Conceptual model of the main resources regarding the transportation module of CityGML

5.2.1.1. Case study using semantic requests

The following CityGML dataset contains a high-detailed street setting in Frankfurt and also five textured buildings in LOD 3 (Figure 5-21).

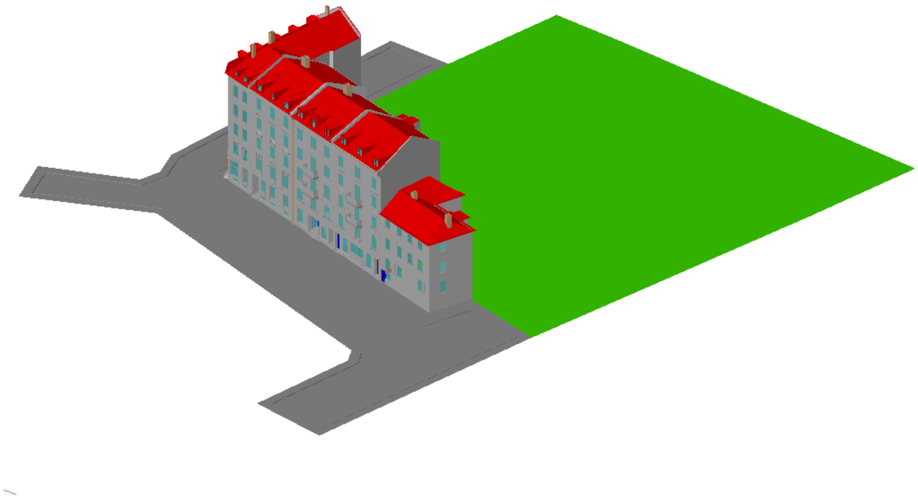
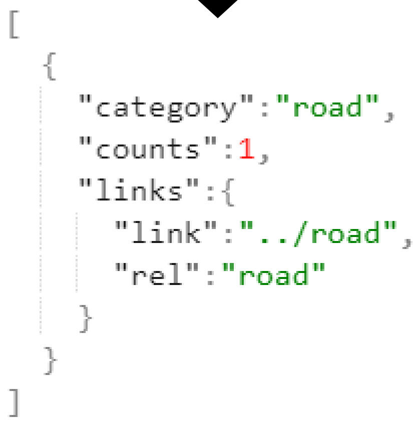


Figure 5-21: CityGML model in the Frankfurt area

By focusing on the transportation module of CityGML, the "tran" main resource is requested as follows (Figure 5-22):

```
../tran
```

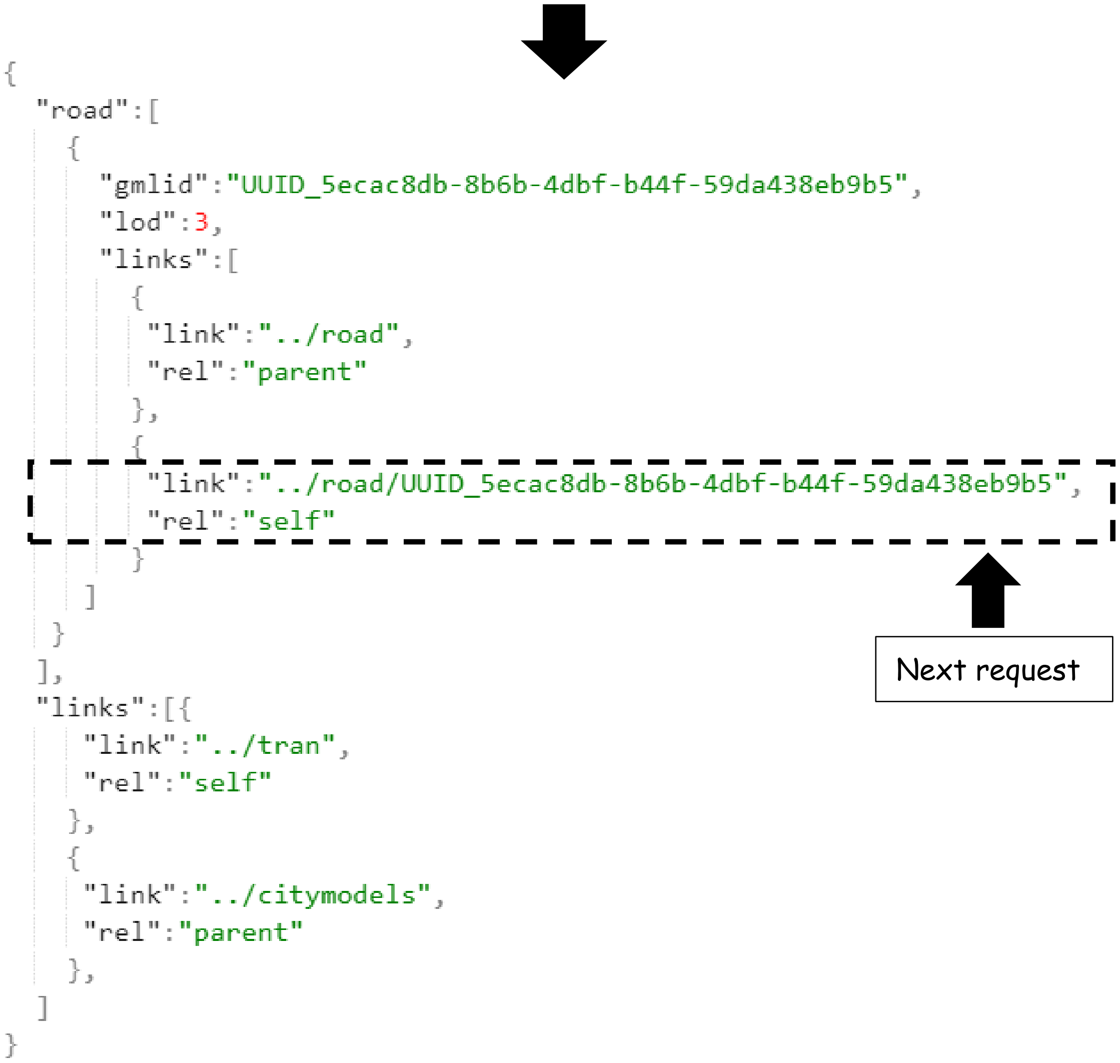


```
[
  {
    "category":"road",
    "counts":1,
    "links":{
      "link":"../road",
      "rel":"road"
    }
  }
]
```

Figure 5-22: "tran" main resource implementation

The above-mentioned request provides information about the availability of the main resources regarding the transportation module such as road, track, railway and square and their respective endpoints. Hence, according to this result, one road is available. Thereafter, since the retrieval of this road requires the corresponding gmlid, the "road" main resource is initially requested.

```
../road
```

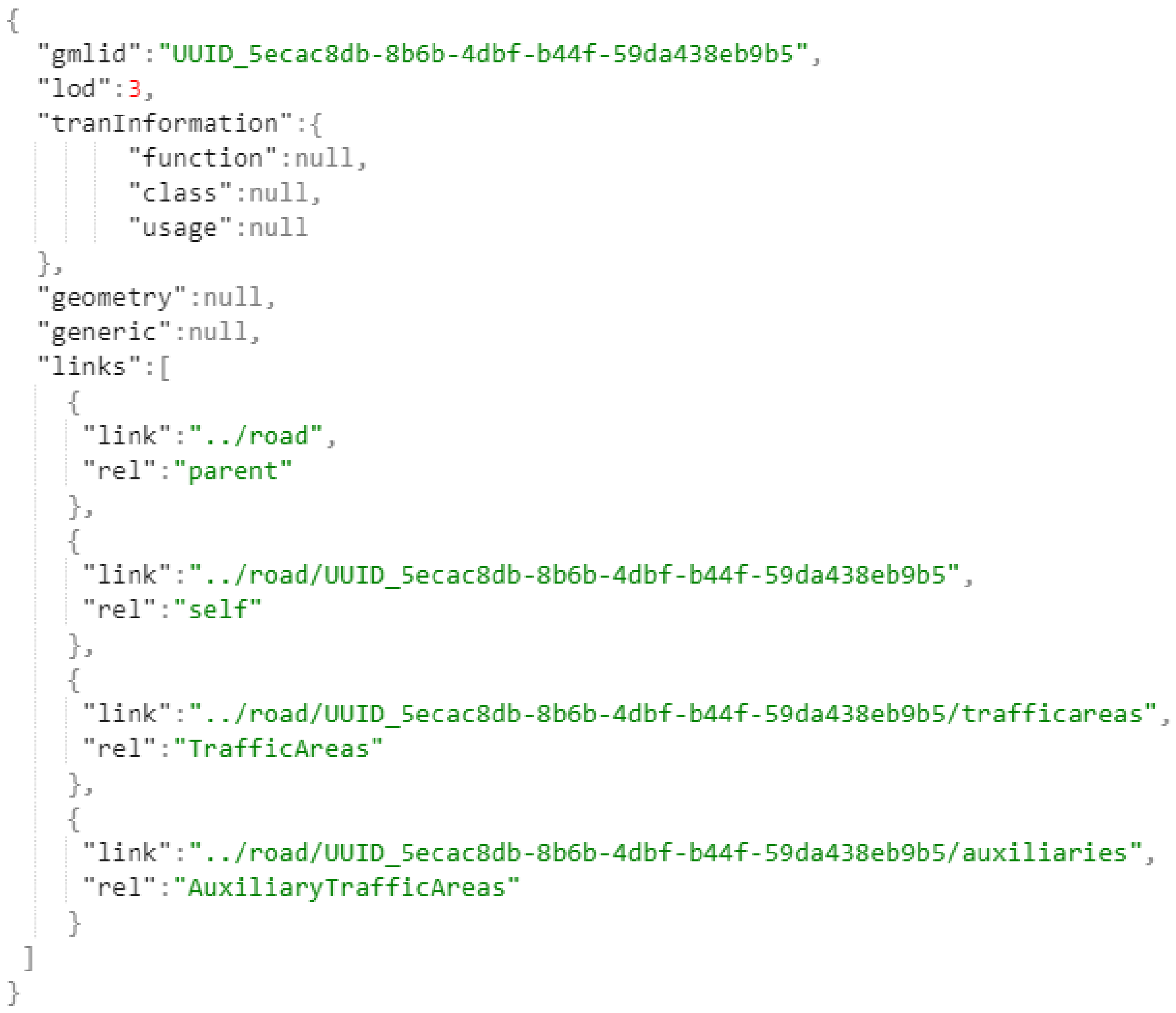


```
{
  "road":[
    {
      "gmlid":"UUID_5ecac8db-8b6b-4dbf-b44f-59da438eb9b5",
      "lod":3,
      "links":[
        {
         "link":"../road",
         "rel":"parent"
        },
        {
         "link":"../road/UUID_5ecac8db-8b6b-4dbf-b44f-59da438eb9b5",
         "rel":"self"
        }
      ]
    }
  ],
  "links":[{
     "link":"../tran",
     "rel":"self"
    },
    {
     "link":"../citymodels",
     "rel":"parent"
    },
  ]
}
```

Figure 5-23: "road" main resource implementation

Next, by using the retrieval gmlid as sub-resource, the specific road is retrieved as well (Figure 5-24).

```
../road/UUID_5ecac8db-8b6b-4dbf-b44f-59da438eb9b5
```



```
{
  "gmlid":"UUID_5ecac8db-8b6b-4dbf-b44f-59da438eb9b5",
  "lod":3,
  "tranInformation":{
        "function":null,
        "class":null,
        "usage":null
  },
  "geometry":null,
  "generic":null,
  "links":[
    {
     "link":"../road",
     "rel":"parent"
    },
    {
     "link":"../road/UUID_5ecac8db-8b6b-4dbf-b44f-59da438eb9b5",
     "rel":"self"
    },
    {
     "link":"../road/UUID_5ecac8db-8b6b-4dbf-b44f-59da438eb9b5/trafficareas",
     "rel":"TrafficAreas"
    },
    {
     "link":"../road/UUID_5ecac8db-8b6b-4dbf-b44f-59da438eb9b5/auxiliaries",
     "rel":"AuxiliaryTrafficAreas"
    }
 ]
}
```

Figure 5-24: Request of specific road

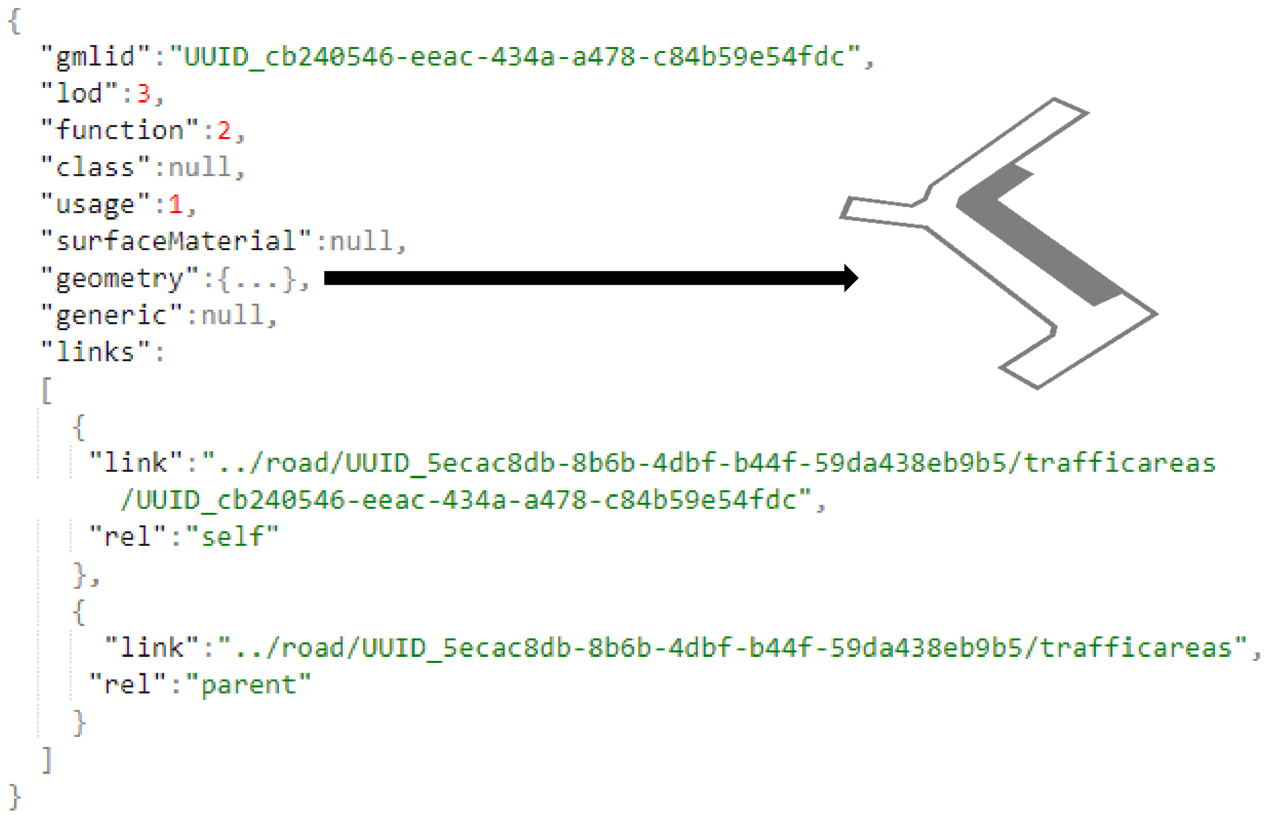
Taking into consideration the above-mentioned result, the said road is further subdivided thematically into "TrafficAreas" and "AuxiliraryTrafficAreas". For that reason, two child sub-resources are provided by the "links" object and also the geometry value of this road is null.

With respect to the "trafficareas" resource, there are two available sub-resources with the following gmlid values:

1) UUID_cb240546-eeac-434a-a478-c84b59e54fdc
2) UUID_c89a449a-62d7-4fc4-9e26-65b24f0c3af1

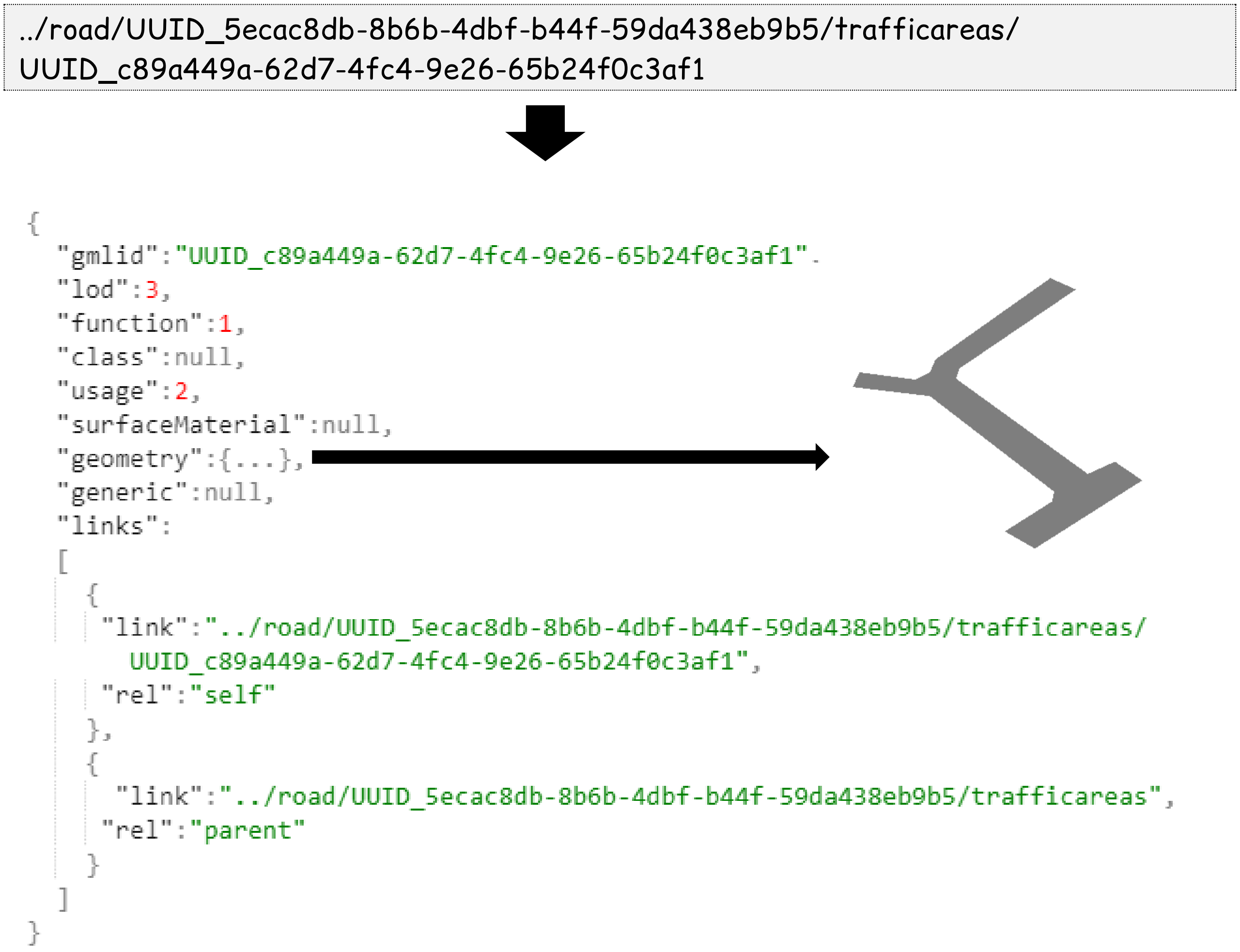
Therefore, by using each of these gmlid values as sub-resource, further information is retrieved (Figure 5-25). More Specific, the first traffic area is a pedestrian area (usage:1), which can be crossed on foot (function:2), while the second one is a driving lane (usage:2), which can be crossed by cars (function:1). It should be noted that for both of these "trafficareas" resources the respective GeoJSON-based geometry is also retrieved.

../road/UUID_5ecac8db-8b6b-4dbf-b44f-59da438eb9b5/trafficareas/
UUID_cb240546-eeac-434a-a478-c84b59e54fdc



```
{
  "gmlid":"UUID_cb240546-eeac-434a-a478-c84b59e54fdc",
  "lod":3,
  "function":2,
  "class":null,
  "usage":1,
  "surfaceMaterial":null,
  "geometry":{...},
  "generic":null,
  "links":
  [
    {
     "link":"../road/UUID_5ecac8db-8b6b-4dbf-b44f-59da438eb9b5/trafficareas
       /UUID_cb240546-eeac-434a-a478-c84b59e54fdc",
     "rel":"self"
    },
    {
      "link":"../road/UUID_5ecac8db-8b6b-4dbf-b44f-59da438eb9b5/trafficareas",
     "rel":"parent"
    }
  ]
}
```

(a)



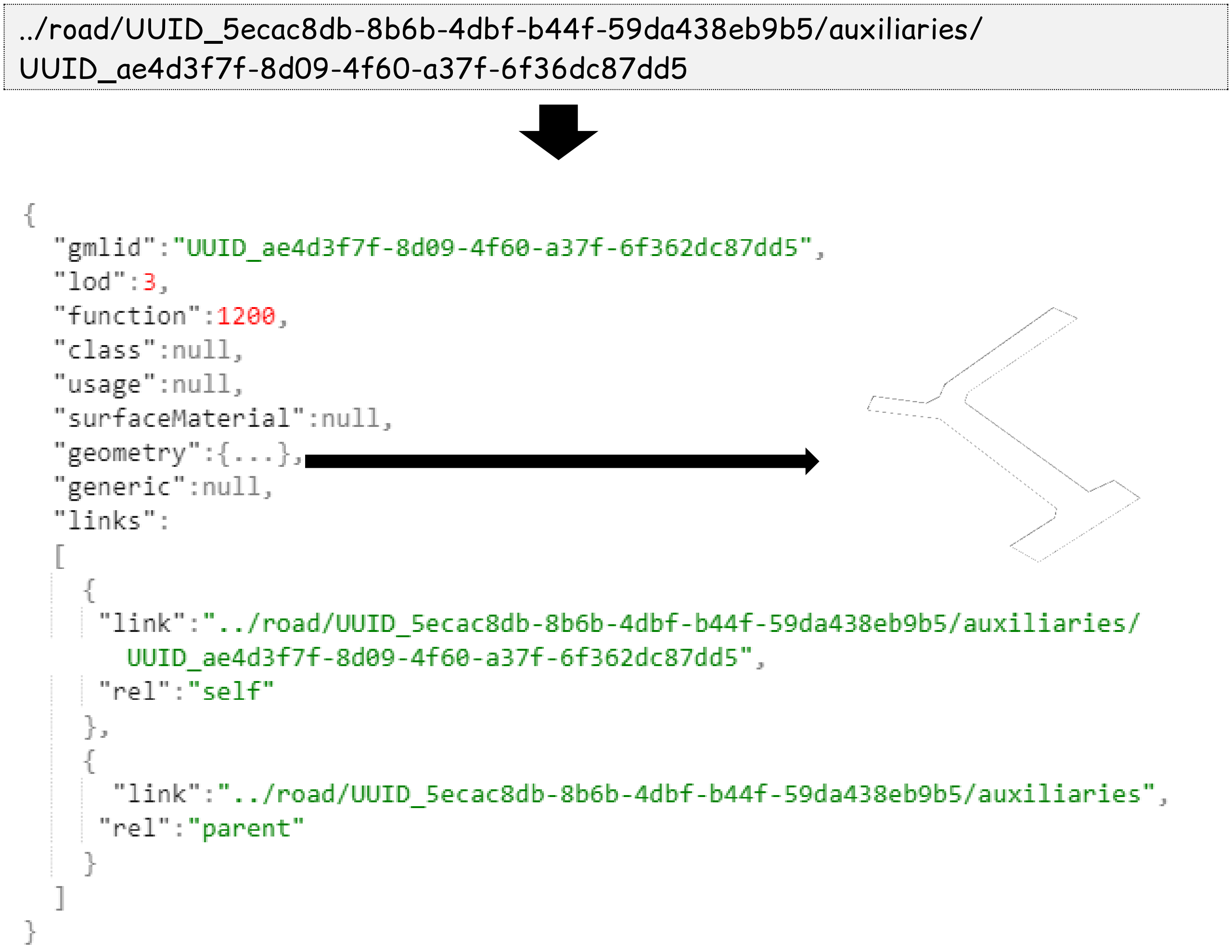
```
../road/UUID_5ecac8db-8b6b-4dbf-b44f-59da438eb9b5/trafficareas/
UUID_c89a449a-62d7-4fc4-9e26-65b24f0c3af1
```

```
{
  "gmlid":"UUID_c89a449a-62d7-4fc4-9e26-65b24f0c3af1",
  "lod":3,
  "function":1,
  "class":null,
  "usage":2,
  "surfaceMaterial":null,
  "geometry":{...},
  "generic":null,
  "links":
  [
    {
     "link":"../road/UUID_5ecac8db-8b6b-4dbf-b44f-59da438eb9b5/trafficareas/
       UUID_c89a449a-62d7-4fc4-9e26-65b24f0c3af1",
     "rel":"self"
    },
    {
      "link":"../road/UUID_5ecac8db-8b6b-4dbf-b44f-59da438eb9b5/trafficareas",
     "rel":"parent"
    }
  ]
}
```

(b)

Figure 5-25: Implementation of "trafficareas" sub-resources

Similarly, Figure 5-26 presents the JSON-based retrieval information derived from the implementation of the respective "auxiliaries" sub-resource. Thus, this area is a kerbstone and is used as a ditch (function: 1200)



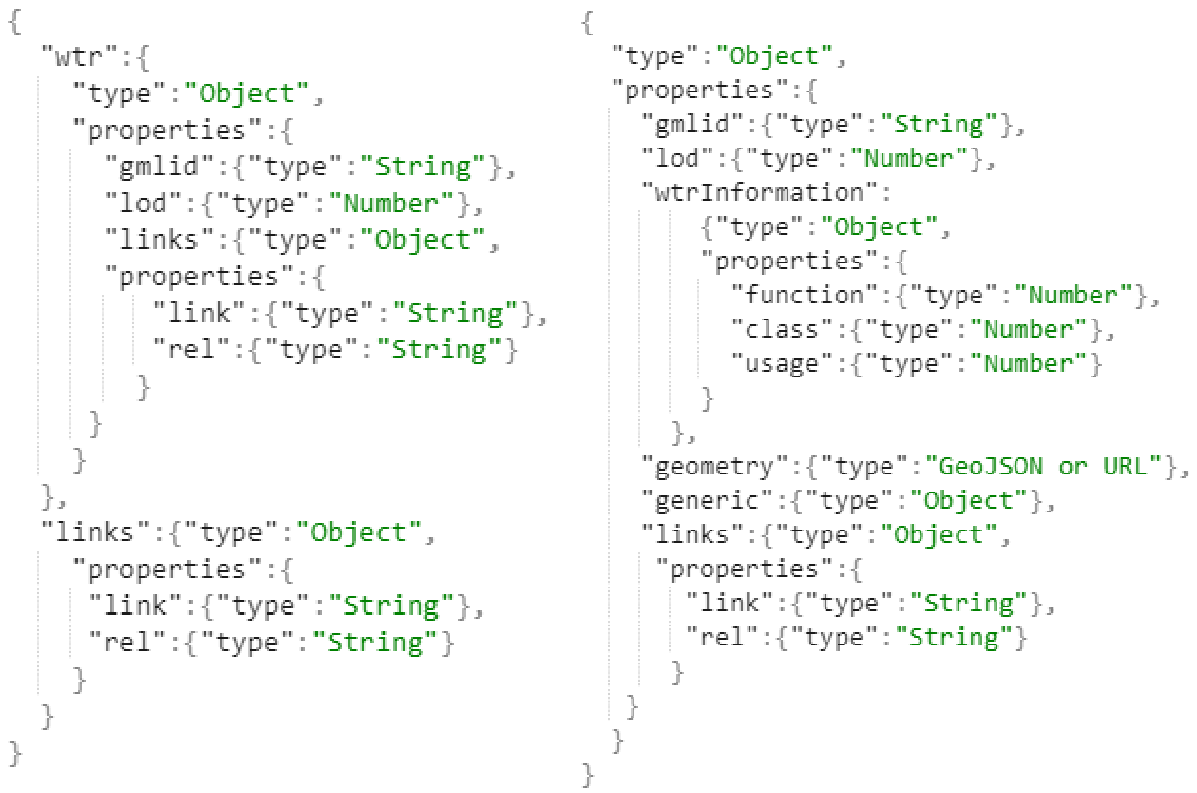
```
../road/UUID_5ecac8db-8b6b-4dbf-b44f-59da438eb9b5/auxiliaries/
UUID_ae4d3f7f-8d09-4f60-a37f-6f36dc87dd5
```

```
{
  "gmlid":"UUID_ae4d3f7f-8d09-4f60-a37f-6f362dc87dd5",
  "lod":3,
  "function":1200,
  "class":null,
  "usage":null,
  "surfaceMaterial":null,
  "geometry":{...},
  "generic":null,
  "links":
  [
    {
     "link":"../road/UUID_5ecac8db-8b6b-4dbf-b44f-59da438eb9b5/auxiliaries/
       UUID_ae4d3f7f-8d09-4f60-a37f-6f362dc87dd5",
     "rel":"self"
    },
    {
      "link":"../road/UUID_5ecac8db-8b6b-4dbf-b44f-59da438eb9b5/auxiliaries",
     "rel":"parent"
    }
  ]
}
```

Figure 5-26: Implementation of "auxiliaries" sub-resource

5.2.2. Wtr resources

Waters have always played a significant role in urbanization process and also, they are considered quite essential for human alimentation and sanitation. With respect to the CityGML v2, a water body model represents the thematic aspects and three-dimensional geometry of rivers, canals, lakes and basins. The retrieval of this model can be achieved using the "wtr" main resource of CityGML RESTful Web service. The retrieval information of this URI is a list of waterbody models and each of these models contains various properties such as lod, wtrInformation, geometry, generic, gmlid and links (Figure 5-27). Also, the general filters (see Table 3-4) can be implemented when the "wtr" resource is requested.



```
{
  "wtr":{
    "type":"Object",
    "properties":{
      "gmlid":{"type":"String"},
      "lod":{"type":"Number"},
      "links":{"type":"Object",
      "properties":{
         "link":{"type":"String"},
         "rel":{"type":"String"}
        }
      }
    }
  },
  "links":{"type":"Object",
    "properties":{
     "link":{"type":"String"},
     "rel":{"type":"String"}
    }
  }
}
```

a) …/wtr

```
{
  "type":"Object",
  "properties":{
    "gmlid":{"type":"String"},
    "lod":{"type":"Number"},
    "wtrInformation":
        {"type":"Object",
        "properties":{
          "function":{"type":"Number"},
          "class":{"type":"Number"},
          "usage":{"type":"Number"}
        }
      },
    "geometry":{"type":"GeoJSON or URL"},
    "generic":{"type":"Object"},
    "links":{"type":"Object",
     "properties":{
       "link":{"type":"String"},
       "rel":{"type":"String"}
      }
    }
  }
}
```

b)…/wtr/{gmlid}

Figure 5-27: JSON-based "wtr" main resource

Similar to the "tran" main resource for both LoD0 and LoD1, there are no extra semantic features and so the only sub-resource is the respective gmlid value. From LoD2 and above the water body is bounded by distinct semantic surfaces such as WaterSurface, which is defined as the boundary between water and air, WaterGroundSurface, which is defined as the boundary between water bodies or between water and underground and WaterClosureSurface, which is the virtual boundary between waterbodies or between water and the end of a modelled region (Figure 5-28).

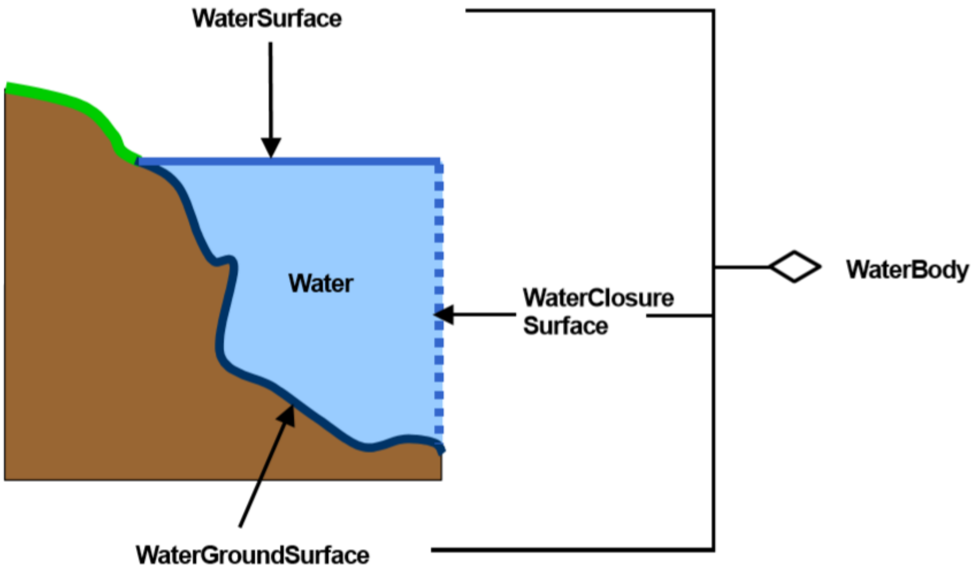


Figure 5-28: Distinct thematic surfaces of the waterbody from LoD2 and above

As a result, the above-mentioned distinct surfaces are the child resources of "wtr" main resource using the following URIs respectively: "water", "grounds" and "closures". It should be noted that the "water" sub-resource does not have the gmlid as child resource, as, according to the WaterBody module of the CityGML v2, a waterbody must have one or zero WaterSurface. Thus, for retrieving the WaterSurface of specific WaterBody the following request should be implemented:

```
../wtr/{gmlid}/water
```

Moreover, only the "water" resource contains extra attribute such as waterLevel, which can be used to describe the water level, for which the given 3D surface geometry was acquired. The said information is especially important when the water body is influenced by the tide. The allowed values of this attribute can be defined in a corresponding code list.

The conceptual design of "wtr" main resource with the respective available information and filters per sub-resource is shown in Figure 5-29. Furthermore, the JSON-based retrieval schemas of the distinct thematic sub-resources are presented in Annex A.8.

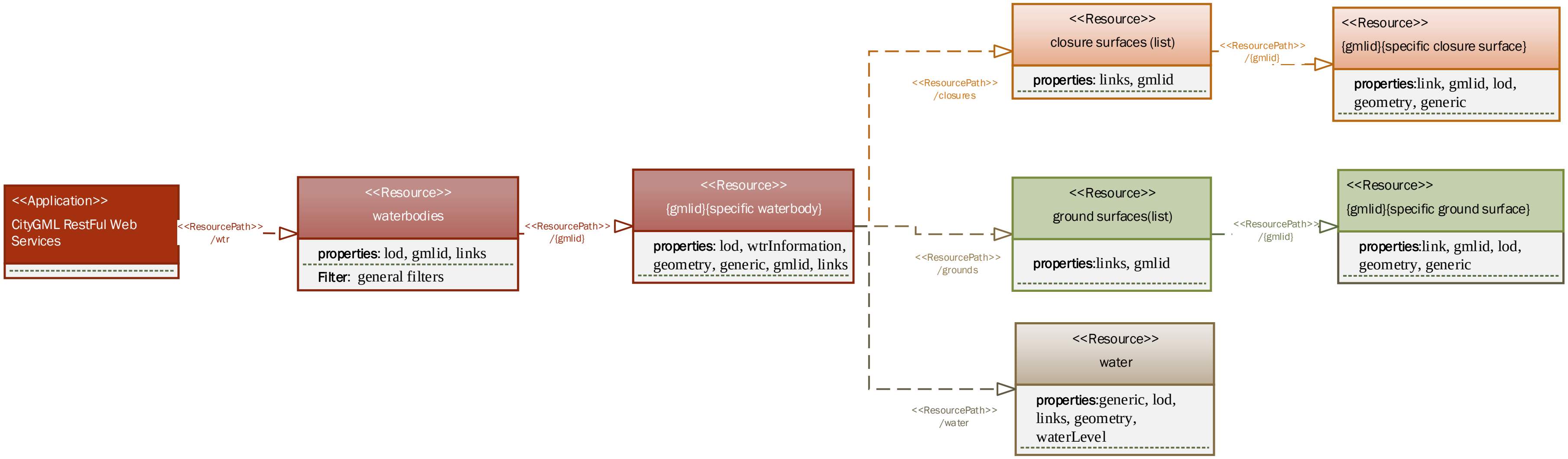

<<Application>>
CityGML RestFul Web Services
<<ResourcePath>> /wtr
<<Resource>>
waterbodies
properties: lod, gmlid, links
Filter: general filters
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid}{specific waterbody}
properties: lod, wtrInformation, geometry, generic, gmlid, links
<<ResourcePath>> /closures
<<Resource>>
closure surfaces (list)
properties: links, gmlid
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid}{specific closure surface}
properties:link, gmlid, lod, geometry, generic
<<ResourcePath>> /grounds
<<Resource>>
ground surfaces(list)
properties:links, gmlid
<<ResourcePath>> /{gmlid}
<<Resource>>
{gmlid}{specific ground surface}
properties:link, gmlid, lod, geometry, generic
<<ResourcePath>> /water
<<Resource>>
water
properties:generic, lod, links, geometry, waterLevel

Figure 5-29: Conceptual design of the "wtr" main resource

6. CONCLUSIONS AND FUTURE WORK

CHAPTER 6: CONCLUSIONS AND FUTURE WORK

6.1. Conclusions

CityGML is considered the optimal standard for the semantic, geometric and topological representation of a city. However, the structure of the CityGML is rather complicated for supporting all these urban complexities. Therefore, retrieving all the available semantic features from this standard is a challenge and the goal of the current dissertation. More specifically, according to the core research question, the CityGML data retrieval should be achieved in relation to key-concepts such as, interoperability, semantic retrieval, easy-to-use and by non-expert.

Initially, the tiled and hierarchically-based approaches for retrieving and visualizing CityGML data using file-based formats, such as X3D, JSON, KML and glTF have been thoroughly investigated, in order to answer the first sub-research question. The visualization of CityGML over the web using the aforementioned 3D graphics, requires the separation of geometric information from semantic information, as they do not have designated place for storing additional object information, which often results in losing rich semantics of CityGML. For that reason, since the OGC I3S and OGC 3D Tiles provide solution to the aforementioned issue by using formats that support the integration of attribute tables, such as SLPK and B3DM, they were further explored. Although the OGC I3S and OGC 3D Tiles provide partial solution, the procedure to generate these files from CityGML source, retrieving all semantic features, is complex, as all of these features must be embedded as data attributes. Last but foremost, the implementation of these solutions is not suitable in terms of interoperability.

Next, taking into account the complex structure of CityGML and the need to retrieve data from distributed sources thus addressing interoperability issues, Web service technologies were investigated. Therefore, the available OGC Geospatial Web services were examined, which, in the context of 3D, are the 3DPS and the WFS. The said research provides answer to the second sub-research question of the current dissertation.

Initially, the 3DPS was examined, as it has been designed to enable the interoperable visualization between various data providers and different browser-based 3D globes and other viewer implementation. The OGC testbed 13 Engineering Report summarizes a proof-of-concept of the use of 3D Tiles and I3S as data delivery formats for the OGC 3DPS interface standard. Hence, the OGC 3DPS standard provides solution to the interoperable portrayal of the 3D city models. However, this portrayal requires complex processing algorithms to convert CityGML into appropriate OGC portrayal standard such as I3S and 3D Tiles. Consequently, the utilization of 3DPS is not the optimal solution for the thesis aim. Therefore, the interoperable and easy-to-use information retrieval of a CityGML based on its semantic characteristics was further examined using the 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 the CityGML schema is much more complex than those usually deployed in WFS. Consequently, the extension of the OGC WFS was further studied and presented. In conclusion, extending WFS to support the retrieval of CityGML data is considered very important. However, the WFS 2.0 and previous version used a Remote-Procedure-Call-Over-HTTP architecture style which was considered state-of-the-art when the WFS standard was originally designed in the late 1990s and early 2000s. Additionally, the WFS, as a query language, enables end-users to submit any type of supported WFS requests and thus difficulties in query optimization can arise. Hence, the integration of the RESTful service architecture on top of WFS was studied in order to steer the end user towards a predefined pattern. In this context, the REST-based architecture was adopted by the upcoming OGC API-Features leaving the Remote-Procedure-Call-Over-HTTP architecture style. The OGC API Features provides basic resources for retrieving features and feature collections. However, the core of OGC API-Features does not currently support the implementation of extra sub-resources, but provides solution for this limitation by extending the Core API by including richer queries from existing OGC standards. Therefore, this implies and requires good knowledge for both the structure of the source (e.g. CityGML) and the respective syntax of the implemented OGC standard. Consequently, the said limitation opposes the fourth sub-research question of this dissertation. Additionally, according to the CityGML architecture, the CityGML structure is more semantic rather than geometric. On the other hand, the OGC WFS is geospatial Web service which means that it was developed with aim of retrieving, visualizing and modifying data based on geometry.

In the next step, the interoperable and easy-to-use information retrieval of a CityGML based on its semantic characteristics was further examined using non- OGC Web Services by focusing on different interoperable approaches. The said research provides answer to the third sub-research question of the current dissertation. Thus, the two types of Web services based on SOAP and REST principle were thoroughly studied and compared. It is concluded that SOAP and REST are two different approaches, with different architectural styles, providing several advantages and disadvantages when compared and so, the architectural decision mostly depends on the specific application. Taking into consideration the complexity of the CityGML structure, the resource-based architecture, which is adopted by the REST, provides an easy-to-use data retrieval mechanism. Hence, the REST-style Web service was chosen. Additionally, the REST was further compared with new state-of-the-art technologies that can be adopted as a CityGML data retrieval mechanism such as GraphQL and Falcor. The implementation of the Falcor or GraphQL presupposes that the client should have good knowledge of either the GraphQL query language or the complex CityGML schema. Additionally, both of these technologies do not currently support geometries and spatial queries. As a result, the REST-based architecture style was finally chosen.

In the final step, a suitable REST-based Web service was designed and the fourth sub-research question of the current dissertation was answered. More specifically, the CityGML RESTful Web service is proposed as the suitable mechanism that meets the requirements of the current dissertation. The utilization of this service for CityGML 2.0 facilitates users to retrieve and manage 3D city models data without presupposing knowledge of the complex structure of CityGML. Also, the resources and sub-resources of this service are based on the ten thematic modules of CityGML 2.0, and their availability depends on the LoDs. So, the sequential retrieval of the semantic features of CityGML is achieved. Additionally, through RESTful implementation, the CityGML RESTful Web Service follows several constraints such as addressability, uniform interface, statelessness, self-describing message and HATEOAS. Therefore, the service interacts by exchanging request and response messages, which contain both the representations of resources and the corresponding metadata. Moreover, the URI of every next request can be retrieved from the “links” object of the current request and so, easy-to-use data retrieval can be completed by non-expert users.

Additionally, CityGML RESTful Web service was conceptually designed to be an information-based retrieval model regarding CityGML 2.0. Therefore, it is not geometrically-based like other OGC standards such as WFS 2.0 and OGC API-Features and thus, the retrieval format is a JSON schema with a list of information. One of this information could be the geometry object, which can be mainly retrieved in GeoJSON format and not GML. The JSON format facilitates the easy-to-use parsing and filtering of the retrieval data by Client-side programming languages such as JavaScript. Thereafter, this data can be further used as input parameters (descriptive or geometric) in spatial analysis tools. It should be noted that the usability of the JSON has led to the creation of the CityJSON format, which provides a simplified alternative to GML encoding of CityGML that is also lightweight and suitable for use on the web and mobile.

In conclusion, the proposed CityGML RESTful Web service is conceptually designed to achieve CityGML data retrieval based on their semantic characteristics by users without any experience and knowledge of the source. So, the core research question of the current dissertation is fully covered by the proposed approach.

As a result, *the optimization of automated retrieval of semantic 3D City Data* is achieved.

6.2. Future Work

6.2.1. OGC standard implementation

Similar to the CityGML RESTful Web service, the REST-based architecture was adopted by the upcoming OGC API-Features leaving the Remote-Procedure-Call-Over HTTP architectural style which is used by previous versions of WFS. The adoption of this architecture style utilizes the WOA and hence, the development of reliable, flexible application is facilitated in an easiest and most economical way (Athanasiou et al., 2018). The OGC API Features provides basic resources for retrieving features and feature collections. These resources are similar to the main resource schema of CityGML RESTful Web Service. However, the core of OGC API-Features does not currently support the implementation of extra sub-resources but provides solution for this limitation by extending the Core API by including richer queries from existing OGC standards. The integration of the sub-resources schema of CityGML RESTful Web Service as an extension to the OGC API-Features will provide a sufficient way to semantically retrieve complex CityGML data. Unfortunately, the aforementioned approach is out of the scope of the OGC API-Features, since the latter is not intended to implement just a standalone API but the same Web API should also implement other standards of the OGC API family (Portele, 2019).

However, during the presentation of the conceptual model of CityGML RESTful Web service at the 3DGeoInfo conference in Singapore, significant positive reviews were received. More specific, the reviewers pointed out the effective solution provided by this approach and also suggested that the CityGML RESTful Web service should be further examined in order to become an OGC standard. Consequently, the aforementioned proposal will be the main future research work. Thereafter, when CityGML RESTful Web service become OGC standard and belong to OGC API family, then it will be able to be implemented by the OGC API Features

6.2.2. Compatibility with the Upcoming version 3 of CityGML for Future Implementation and upgrade

Since January 2018, CityGML v3.0 conceptual model has been made available in development mode on the original GitHub repository for OGC CityGML 3.0. This upcoming version has been fully revised bringing a number of improvements, extensions and new functionalities (Kutzner, Chaturvedi, & Kolbe, 2020) to reflect the increasing need for better interoperability with other relevant standards in the field like IFC, IndoorGML, Land Administrator Domain Model (LADM) and INSPIRE. The architecture of the CityGML 3 including the new additions is presented in Figure 6-1.

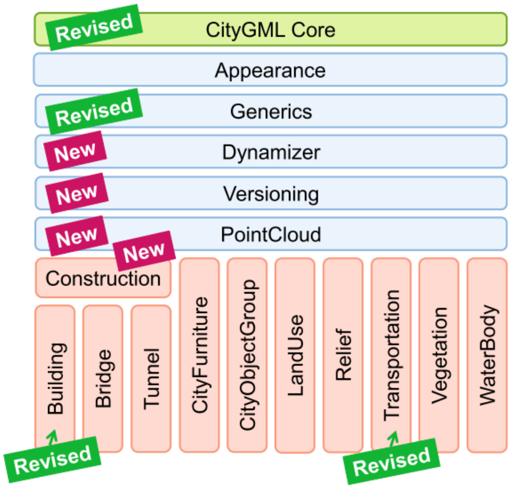


Figure 6-1: CityGML 3.0 modules overview

(Kutzner, Chaturvedi, & Kolbe, 2020)

More specific, all modules from CityGML 2.0 will be part of CityGML 3.0. In addition, the new modules Dynamizer, Versioning, PointCloud and Construction will be introduced, and the modules Core, Generic, Building, and Transportation will be revised. These changes are briefly presented in following paragraphs focusing on their impact on the conceptual model of CityGML RESTful Web service.

6.2.2.1. Revised Core Module

In CityGML 3.0, a clear semantic distinction of spatial features is introduced by mapping all city objects onto the semantic concepts of spaces and space boundaries. A space is an entity of volumetric extent in the real word. So, since the Buildings, water bodies, trees, rooms and traffic spaces have a volumetric extent, they are modelled as spaces. However, the space is further subdivided into "physical spaces" and "logical spaces". The first one refers to the spaces that are fully or partially bounded by physical objects. The "physical spaces" is an abstract class, which is mainly used to separate the physical from the logical space. Hence, it does not affect the conceptual design of the main resources of CityGML RESTful Web service.

On the other hand, logical spaces are spaces that are not necessarily bounded by physical objects, but are defined according to thematic considerations and so, they can also be bounded by non-physical or represent aggregation of physical spaces. For instance, a building unit is a logical space as it aggregates specific rooms to flats. As a result, this sub-category of space should be considered by the CityGML RESTful Web service.

With regard to the space boundary, it does not affect the conceptual design of the CityGML RESTful Web service sub-resources as it is mainly used as an abstract class for the boundary surfaces such as "WallSurface", "RoofSurface" etc.

CityGML 3.0 will include a revised LoD concept which comprises a central definition of all geometries in the Core module and the representation of the interior of city objects at any level of detail. More specific, the LoD concept is modified, based on the proposed LoDs as described by Lowner, et al. (2016). According to the authors, the main barrier in the current concept of LoD is that the interior structure of an element can only be represented if the exterior shell is represented in LoD4, which implies the highest semantic complexity and geometric detail. Therefore, in CityGML 3.0, LoD4 is replaced by LoD0 to LoD3 for exterior and indoor objects and all feature types can be represented in each LoD. So, it is possible to model the outside shell of a model in LoD1 while representing the interior structure in LoD2 or LoD3. It should be noted that the main structure of the CityGML RESTful Web service is not affected by this important change as its conceptual model of the resources is designed by taking into account the semantic aspect of CityGML. However, the availability of the sub-resources should be modified so that these resources can be provided based on the new concept of LoD.

6.2.2.2. New Construction module

The Construction module groups all classes which are similar over different types of constructions like buildings, tunnels, bridges and introduces a new class "OtherConstruction" to represent other man-made structures not belonging to any of the aforementioned three modules (e.g. large chimneys or city walls). More specific, the construction elements refer to the boundary and opening surfaces regarding the modules building, bridge and tunnel which remain the same even if they belong to the construction module. So, the respective boundary and opening child resources of the CityGML RESTful Web service will not be changed.

6.2.2.3. New Versioning module

The Versioning module introduces bitemporal timestamps for all objects. Therefore, except of the attributes "creationDate and "terminalDate" from CityGML 2.0, all objects now can have a second lifespan expressed by the attributes "validFrom" and "validTo". Additionally, each geographic feature will have two identifiers such as "identifies" and "gml:id". The value of the "identifier" property will be stable along the lifetime of the real-word object, while the "gml:id" attribute will be constructed from the "identifier" with concatenated timestamp (Chaturvedi, et al., 2017). The versioning module could be supported by the CityGML RESTful Web service by defining a new object "versioning" as an information retrieval for each resource. Additionally, this object should be included in the general filters. The object "versioning" will be a List of key value pairs based on the Versioning module of the CityGML 3.0.

6.2.2.4. New Dynamizer module

The Dynamizer module improves the usability of CityGML for different kinds of simulations and also facilitates the integration of sensors with 3D city models. Through the Dynamizers, the link of timeseries data (OGC TimeseriesML, OGC observation and Measurement, tabulated data in external files like CSV) to a specific attribute or property of a specific object within the 3D city model will be achieved (Chaturvedi and Kolbe, 2017). This capability facilitates the dynamic or real time updating of the source data and it can be implemented by the CityGML RESTful Web service similar to the "Dynamizer" ADE resource (see 3.2.2.1)

6.2.2.5. New PointCloud module

The thematic surfaces can also be provided by 3D point clouds using MultiPoint geometry. This capability does not affect the retrieval result schema of the CityGML RESTful Web service as the retrieval result will be retrieved in GeoJSON-based format.

6.2.2.6. The revised transportation module

In the new Transportation module of CityGML 3.0, the transportation objects such as road, track, railway and square, can be subdivided into sections. These sections can be regular road, track or railway legs, intersection areas or roundabouts, each belonging to multiple Road or Track objects. Thereafter, in order to avoid a redundant representation of this shared object, Xlinks will be used in the CityGML 3.0 instance document to reference the shared section (Beil & Kolbe, 2017). Additionally, "TrafficSpace" and "AuxilliaryTrafficSpace" will be introduced in addition to "TrafficArea" and "AuxilliaryTrafficArea" of CityGML 2.0. Also, the traffic space can have and optional "ClearanceSpace". Moreover, new semantic surface will be integrated such as "Hole" and "HoleSurface". As a result, in CityGML 3.0, the Transportation Objects will have an areal as well as center line representation for each LoD and, in addition, extra semantic surfaces will be introduced. Consequently, taking into account all the above-mentioned changes, the sub-resources of the "tun" main resource should be modified in future research work.

6.2.2.7. The components of Building module

The new Building module mainly remains the same. However, two new subdivision will be included as logical spaces such as "BuildingUnit" and "Storey". These subdivisions will have Xlinks to the respective rooms. So, two new child resources of "bldg" resource should be embedded such as "buildingunits" and "storeys". These resources will retrieve a list of the respective building units and storeys. Thereafter, the retrieval of specific object is achieved by implementing the respective gmlid as sub-resource. This sub-resource will contain a list of links to the corresponding “rooms” sub-resources that includes. The implementation of the new sub-resources could be the following:

```
/bldg/{gmlid}/{buildingunits or storeys}/{gmlid}
```

ANNEX A: JSON-BASED SCHEMA OF SUB-RESOURCES

ANNEX A: JSON-BASED SCHEMA OF SUB-RESOURCES

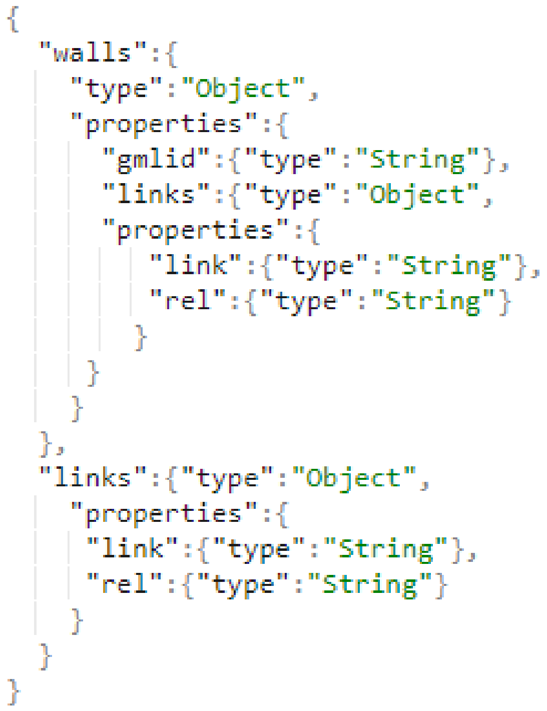
ANNEX A: JSON-BASED SCHEMA OF SUB-RESOURCES

ANNEX A: JSON-BASED SCHEMA OF SUB-RESOURCES

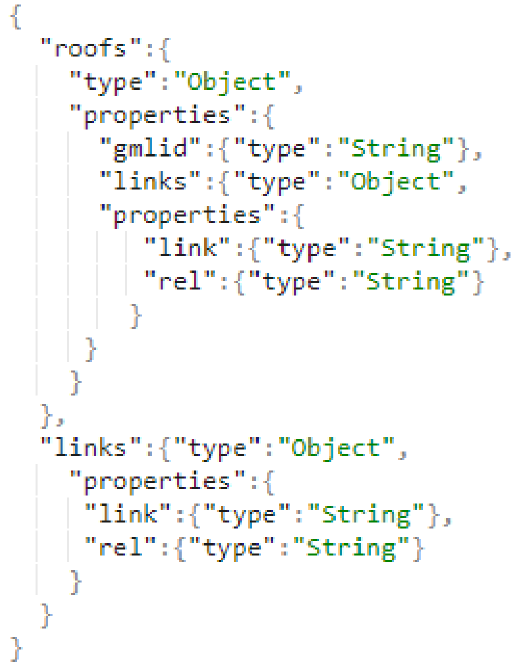
A.1. Boundary Surface Resources

Exterior: ../{main resource}/{gmlid}/walls
Interior: .. /{main resource}/{gmlid}/{interior spaces}/{gmlid}/walls
*rooms or hollowspaces

Exterior: ../{main resource }/{gmlid}/roofs



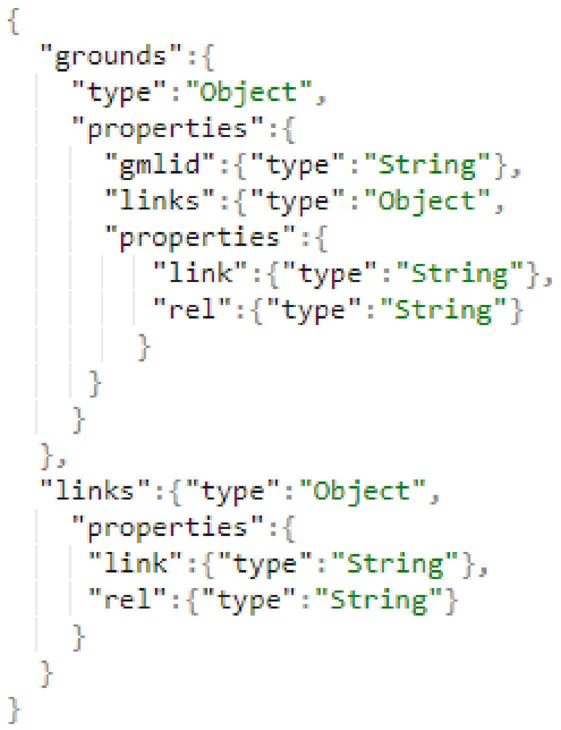
```
{
  "walls":{
    "type":"Object",
    "properties":{
      "gmlid":{"type":"String"},
      "links":{"type":"Object",
      "properties":{
         "link":{"type":"String"},
         "rel":{"type":"String"}
        }
     }
    }
  },
  "links":{"type":"Object",
    "properties":{
     "link":{"type":"String"},
     "rel":{"type":"String"}
    }
  }
}
```



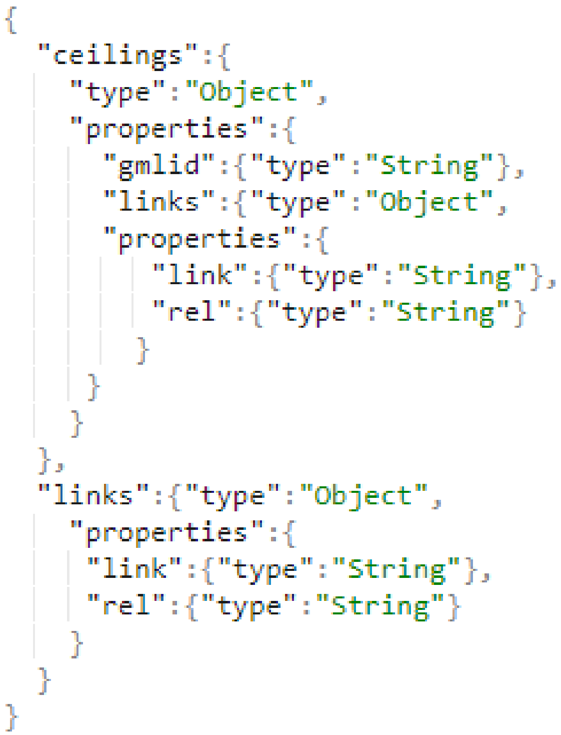
```
{
  "roofs":{
    "type":"Object",
    "properties":{
      "gmlid":{"type":"String"},
      "links":{"type":"Object",
      "properties":{
         "link":{"type":"String"},
         "rel":{"type":"String"}
        }
     }
    }
  },
  "links":{"type":"Object",
    "properties":{
     "link":{"type":"String"},
     "rel":{"type":"String"}
    }
  }
}
```

Exterior: .. /{main resource} /{gmlid}/grounds

Exterior: .. /{main resource} /{gmlid}/ceilings
Interior: .. /{main resource} /{gmlid}/{interior space*}/{/{gmlid}/ceilings
*rooms or hollowspaces

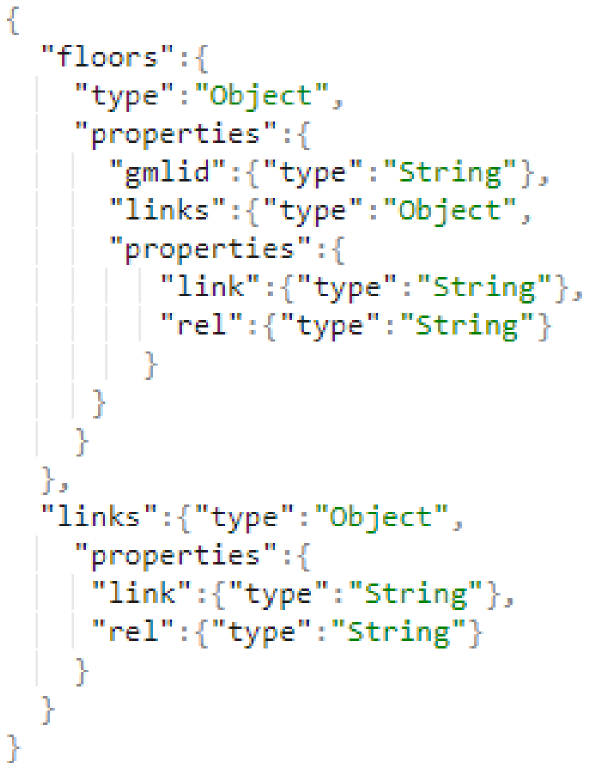


```
{
  "grounds":{
    "type":"Object",
    "properties":{
      "gmlid":{"type":"String"},
      "links":{"type":"Object",
      "properties":{
         "link":{"type":"String"},
         "rel":{"type":"String"}
        }
      }
    }
  },
  "links":{"type":"Object",
    "properties":{
     "link":{"type":"String"},
     "rel":{"type":"String"}
    }
  }
}
```



```
{
  "ceilings":{
    "type":"Object",
    "properties":{
      "gmlid":{"type":"String"},
      "links":{"type":"Object",
      "properties":{
         "link":{"type":"String"},
         "rel":{"type":"String"}
        }
      }
    }
  },
  "links":{"type":"Object",
    "properties":{
     "link":{"type":"String"},
     "rel":{"type":"String"}
    }
  }
}
```

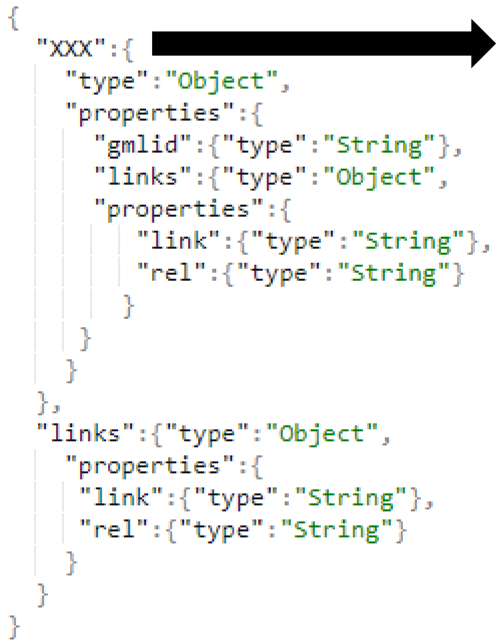
Exterior: .. /{main resource} /{gmlid}/floors
Interior: .. /{main resource} /{gmlid}/{interior space*}/{gmlid}/floors
*rooms or hollowspaces



```
{
  "floors":{
    "type":"Object",
    "properties":{
      "gmlid":{"type":"String"},
      "links":{"type":"Object",
      "properties":{
         "link":{"type":"String"},
         "rel":{"type":"String"}
        }
      }
    }
  },
  "links":{"type":"Object",
    "properties":{
     "link":{"type":"String"},
     "rel":{"type":"String"}
    }
  }
}
```

A.2. Installation Resources

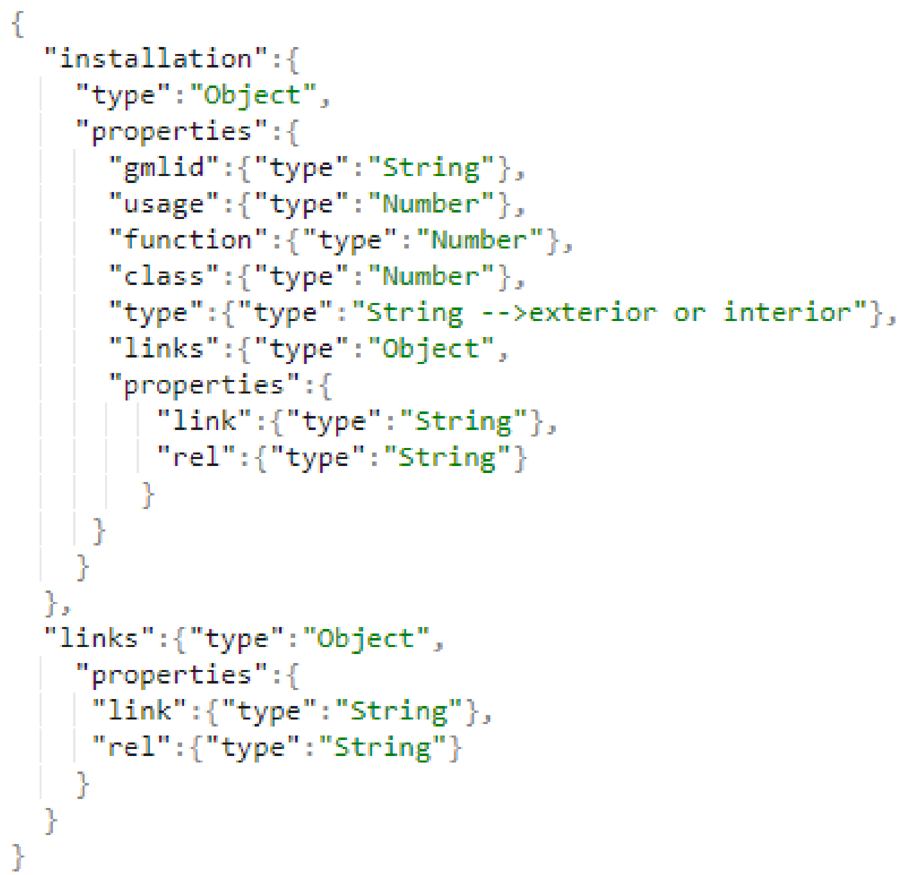
Exterior: .. /{main resource} /{gmlid}/installation
Interior: (a) .. {main resource} /{gmlid}/installation
(b) .. {main resource} /{gmlid}/{interior space *} /{gmlid}/installation
* rooms or hollowspaces



```
{
  "XXX":{
    "type":"Object",
    "properties":{
      "gmlid":{"type":"String"},
      "links":{"type":"Object",
      "properties":{
         "link":{"type":"String"},
         "rel":{"type":"String"}
        }
     }
    }
  },
  "links":{"type":"Object",
    "properties":{
     "link":{"type":"String"},
     "rel":{"type":"String"}
    }
  }
}
```

The character "XXX" is defined based on the exterior boudary surfaces such as walls, roofs, grounds, ceilings and floors. Also, the retrieval data is array of features that belongs to the corresponding surface.

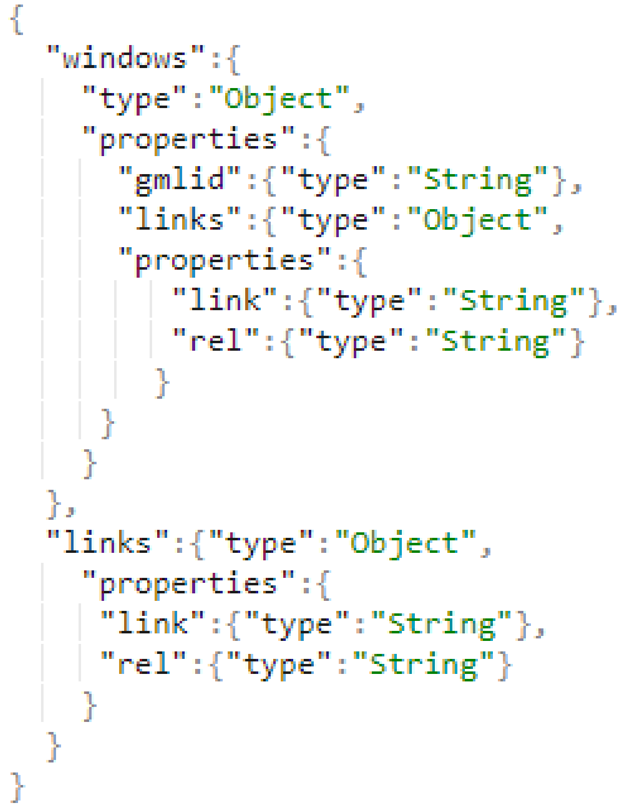
Exterior: .. /{main resource} /{gmlid}/installation/{gmlid}
Interior: (a) .. {main resource} /{gmlid}/installation/{gmlid}
(b) .. {main resource} /{gmlid}/{interior space *} /{gmlid}/installation/{gmlid}
* rooms or hollowspaces



```
{
  "installation":{
    "type":"Object",
    "properties":{
      "gmlid":{"type":"String"},
      "usage":{"type":"Number"},
      "function":{"type":"Number"},
      "class":{"type":"Number"},
      "type":{"type":"String -->exterior or interior"},
      "links":{"type":"Object",
      "properties":{
         "link":{"type":"String"},
         "rel":{"type":"String"}
        }
      }
    }
  },
  "links":{"type":"Object",
    "properties":{
     "link":{"type":"String"},
     "rel":{"type":"String"}
    }
  }
}
```

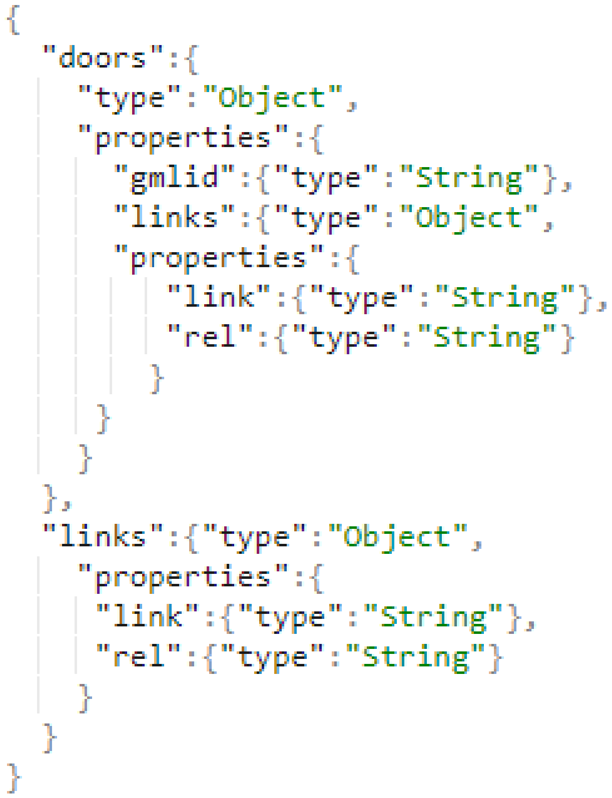
A.3. Opening Resources

Exterior: ../ {main resource} /{gmlid}/{ boundary surfaces [2] }/{gmlid}/windows
Interior: ../ {main resource} /{gmlid}/{interior spaces [1] } /{gmlid}/{boundary surfaces [2] }/{gmlid}/windows
[1] rooms or hollowspaces
[2] walls or roofs



```
{
  "windows":{
    "type":"Object",
    "properties":{
      "gmlid":{"type":"String"},
      "links":{"type":"Object",
      "properties":{
         "link":{"type":"String"},
         "rel":{"type":"String"}
        }
      }
    }
  },
  "links":{"type":"Object",
    "properties":{
     "link":{"type":"String"},
     "rel":{"type":"String"}
    }
  }
}
```

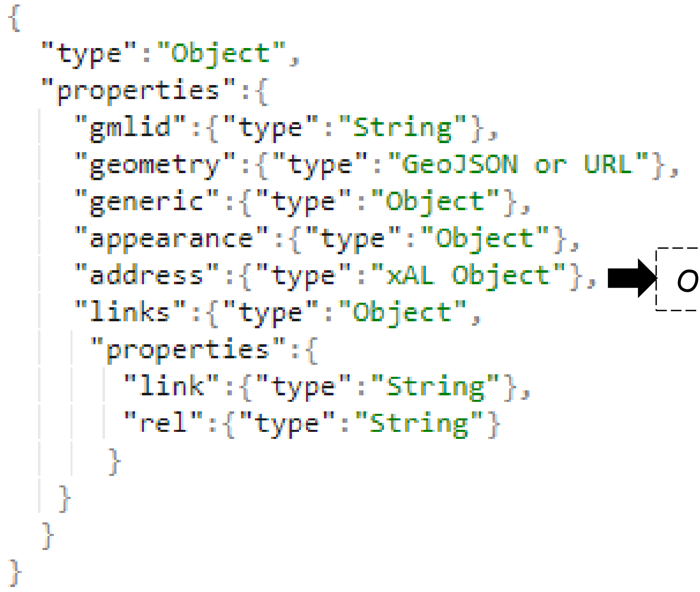
Exterior: ../ {main resource} /{gmlid}/{ boundary surfaces [2] }/{gmlid}/doors
Interior: ../ {main resource} /{gmlid}/{interior spaces [1] } /{gmlid}/{boundary surfaces [2] }/{gmlid}/doors
[1] rooms or hollowspaces
[2] walls or roofs



```
{
  "doors":{
    "type":"Object",
    "properties":{
      "gmlid":{"type":"String"},
      "links":{"type":"Object",
      "properties":{
         "link":{"type":"String"},
         "rel":{"type":"String"}
        }
      }
    }
  },
  "links":{"type":"Object",
    "properties":{
     "link":{"type":"String"},
     "rel":{"type":"String"}
    }
  }
}
```

Exterior: ../ {main resource} /{gmlid}/{ boundary surfaces[2] }/{gmlid}/{opening [3]}/{gmlid}
Interior: ../ {main resource} /{gmlid}/{interior spaces[1] } /{gmlid}/{boundary surfaces[2] }/{gmlid}/ {opening [3]}/{gmlid}

[1] rooms or hollowspaces
[2] walls or roofs
[3] windows or doors

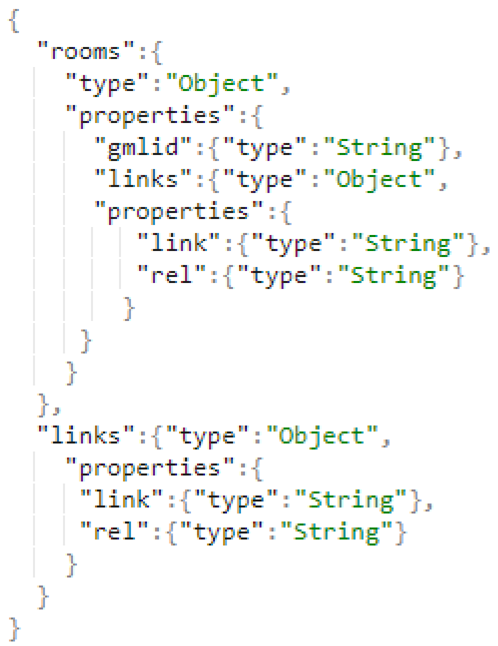


```
{
  "type":"Object",
  "properties":{
    "gmlid":{"type":"String"},
    "geometry":{"type":"GeoJSON or URL"},
    "generic":{"type":"Object"},
    "appearance":{"type":"Object"},
    "address":{"type":"xAL Object"},
    "links":{"type":"Object",
     "properties":{
       "link":{"type":"String"},
       "rel":{"type":"String"}
     }
   }
  }
}
```

Only available for door opening resource

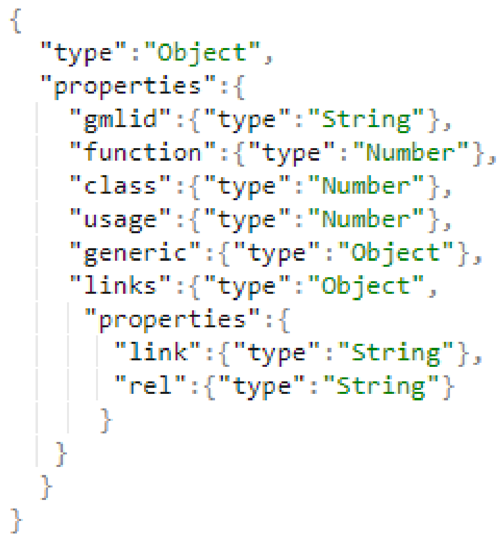
A.4. "rooms" Resources

Interior: .. /{main resource} /{gmlid}/rooms



```
{
  "rooms":{
    "type":"Object",
    "properties":{
      "gmlid":{"type":"String"},
      "links":{"type":"Object",
      "properties":{
         "link":{"type":"String"},
         "rel":{"type":"String"}
        }
     }
    }
  },
  "links":{"type":"Object",
    "properties":{
     "link":{"type":"String"},
     "rel":{"type":"String"}
    }
  }
}
```

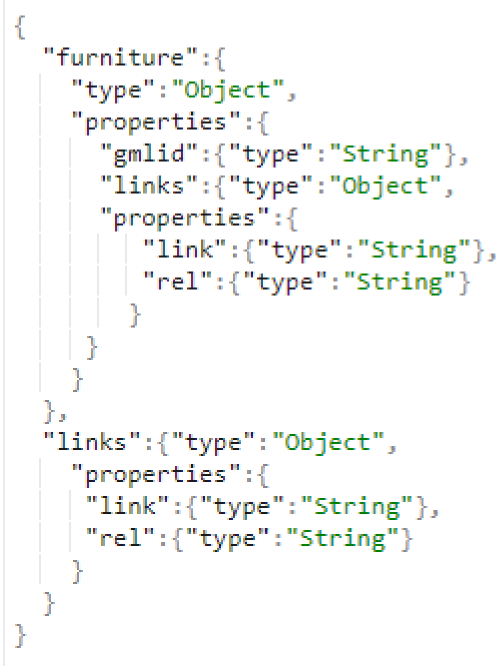
Interior: .. {main resource} /{gmlid}/rooms/{gmlid}



```
{
  "type":"Object",
  "properties":{
    "gmlid":{"type":"String"},
    "function":{"type":"Number"},
    "class":{"type":"Number"},
    "usage":{"type":"Number"},
    "generic":{"type":"Object"},
    "links":{"type":"Object",
     "properties":{
       "link":{"type":"String"},
       "rel":{"type":"String"}
      }
   }
  }
}
```

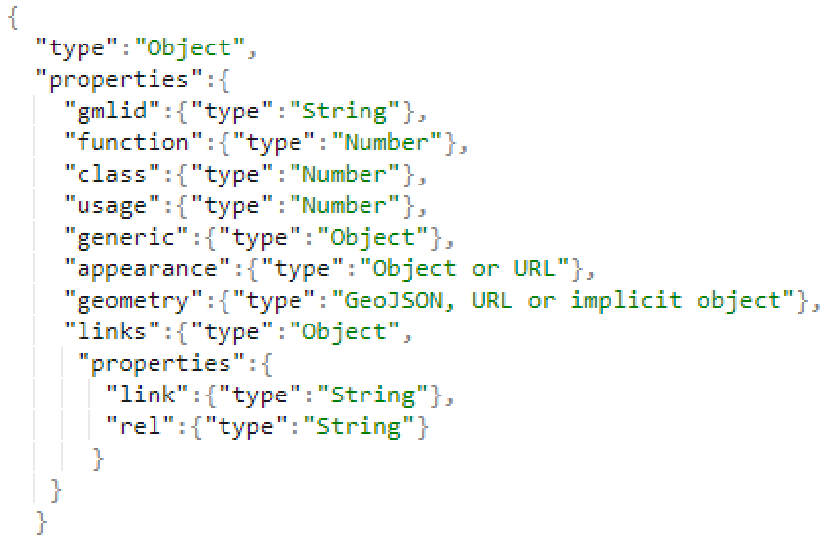
A.5. "furniture" Resource

Interior: .. /{main resource} /{gmlid}/{ interior space *}/{gmlid}/furniture
* rooms or hollowspaces



```
{
  "furniture":{
    "type":"Object",
    "properties":{
      "gmlid":{"type":"String"},
      "links":{"type":"Object",
      "properties":{
         "link":{"type":"String"},
         "rel":{"type":"String"}
        }
      }
    }
  },
  "links":{"type":"Object",
    "properties":{
     "link":{"type":"String"},
     "rel":{"type":"String"}
    }
  }
}
```

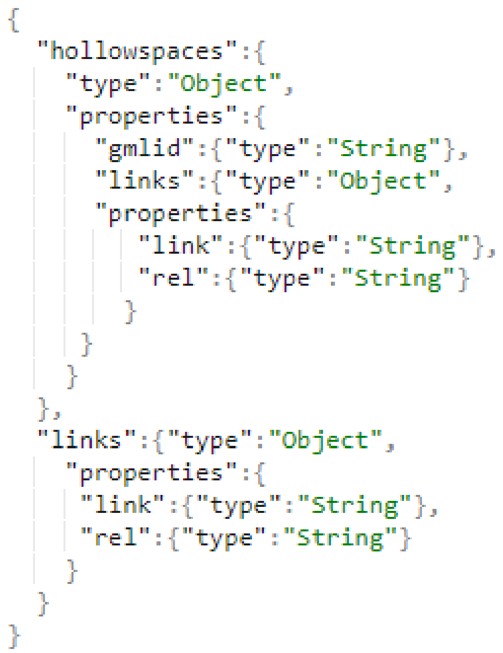
Interior: ../{main resource} /{gmlid}/{ interior space *}/{gmlid}/furniture/{gmlid}
* rooms or hollowspaces



```
{
  "type":"Object",
  "properties":{
    "gmlid":{"type":"String"},
    "function":{"type":"Number"},
    "class":{"type":"Number"},
    "usage":{"type":"Number"},
    "generic":{"type":"Object"},
    "appearance":{"type":"Object or URL"},
    "geometry":{"type":"GeoJSON, URL or implicit object"},
    "links":{"type":"Object",
     "properties":{
       "link":{"type":"String"},
       "rel":{"type":"String"}
      }
    }
  }
```

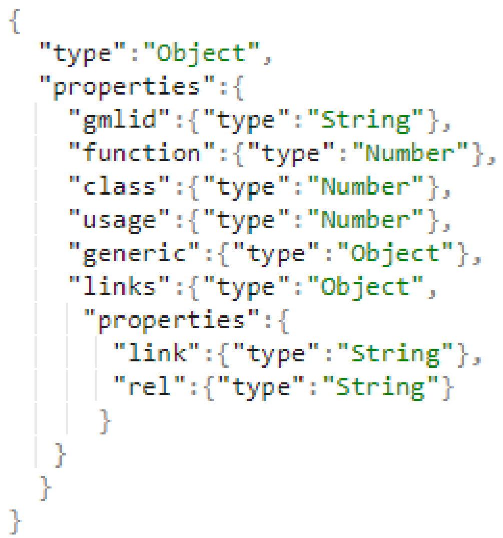
A.6. "hollowspaces" Resources

Interior: .. /tun /{gmlid}/hollowspaces



```
{
  "hollowspaces":{
    "type":"Object",
    "properties":{
      "gmlid":{"type":"String"},
      "links":{"type":"Object",
      "properties":{
         "link":{"type":"String"},
         "rel":{"type":"String"}
        }
      }
    }
  },
  "links":{"type":"Object",
    "properties":{
     "link":{"type":"String"},
     "rel":{"type":"String"}
    }
  }
}
```

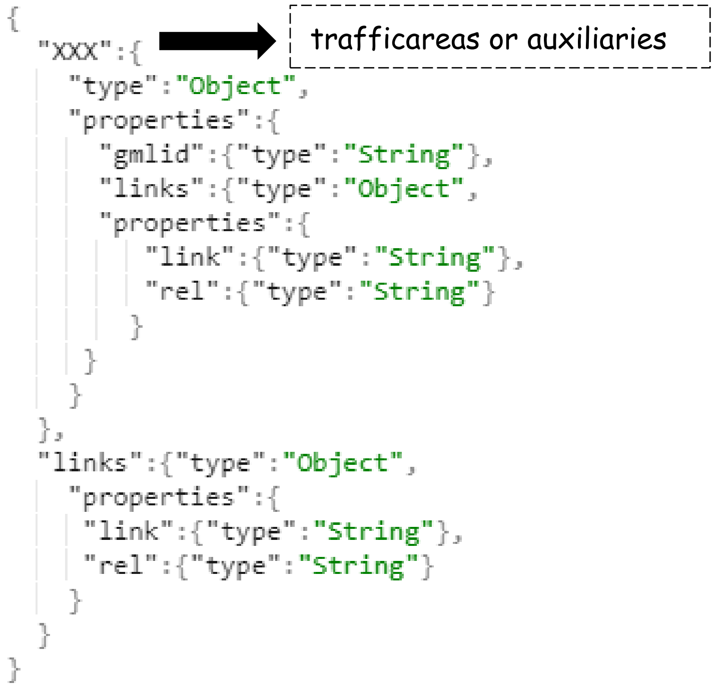
Interior: ../tun /{gmlid}/hollowspaces/{gmlid}



```
{
  "type":"Object",
  "properties":{
    "gmlid":{"type":"String"},
    "function":{"type":"Number"},
    "class":{"type":"Number"},
    "usage":{"type":"Number"},
    "generic":{"type":"Object"},
    "links":{"type":"Object",
     "properties":{
       "link":{"type":"String"},
       "rel":{"type":"String"}
      }
    }
  }
}
```

A.7. "trafficareas" and "auxiliaries" Resources

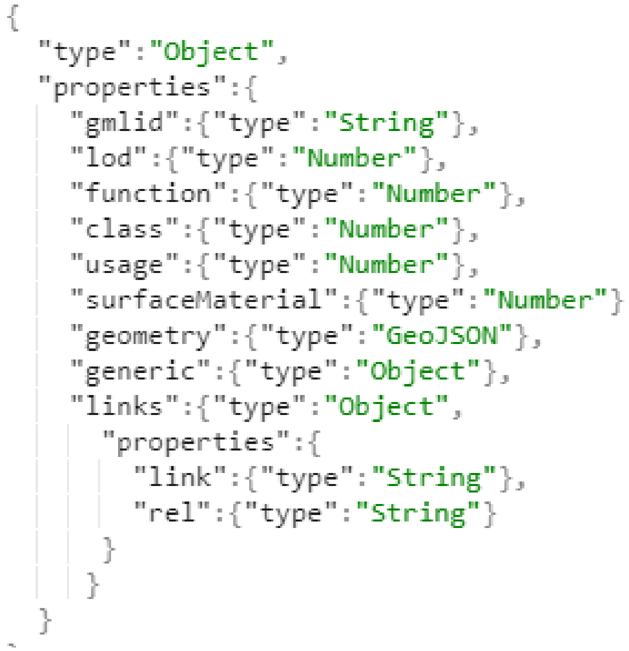
Exterior: ../{extra main resource* }/{gmlid}/{trafficareas or auxiliaries}
*road, track, railway, square



trafficareas or auxiliaries

```
{
  "XXX":{
    "type":"Object",
    "properties":{
      "gmlid":{"type":"String"},
      "links":{"type":"Object",
      "properties":{
          "link":{"type":"String"},
          "rel":{"type":"String"}
        }
      }
    }
  },
  "links":{"type":"Object",
    "properties":{
     "link":{"type":"String"},
     "rel":{"type":"String"}
    }
  }
}
```

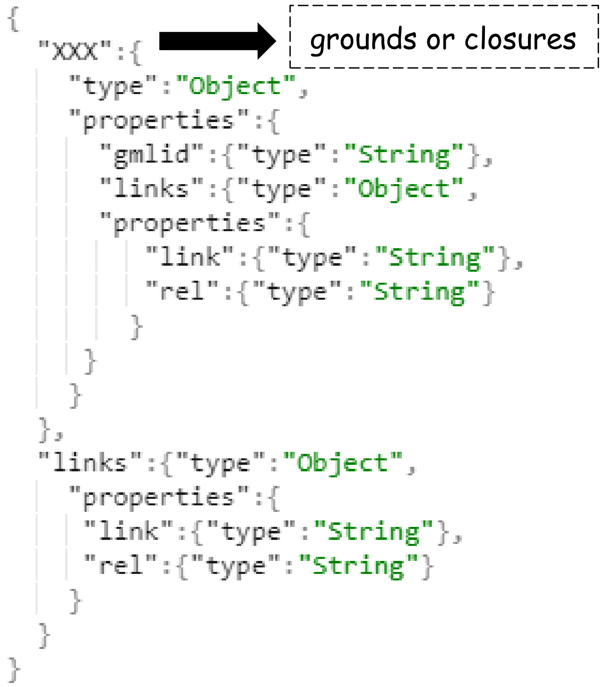
Exterior: ../{extra main resource* }/{gmlid}/{trafficareas or auxiliaries}/{gmlid}
*road, track, railway, square



```
{
  "type":"Object",
  "properties":{
    "gmlid":{"type":"String"},
    "lod":{"type":"Number"},
    "function":{"type":"Number"},
    "class":{"type":"Number"},
    "usage":{"type":"Number"},
    "surfaceMaterial":{"type":"Number"}
    "geometry":{"type":"GeoJSON"},
    "generic":{"type":"Object"},
    "links":{"type":"Object",
      "properties":{
        "link":{"type":"String"},
        "rel":{"type":"String"}
      }
    }
  }
```

A.8. "water", "grounds" and "closures" Resources

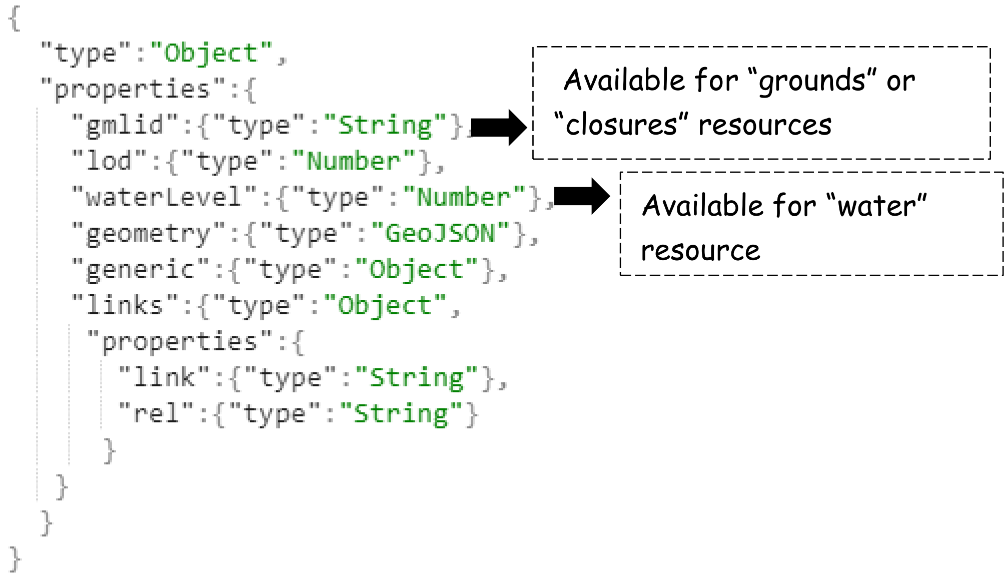
Exterior: ../wtr/{gmlid}/{grounds or closures}



```
{
  "XXX":{
    "type":"Object",
    "properties":{
      "gmlid":{"type":"String"},
      "links":{"type":"Object",
      "properties":{
         "link":{"type":"String"},
         "rel":{"type":"String"}
        }
      }
    }
  },
  "links":{"type":"Object",
    "properties":{
     "link":{"type":"String"},
     "rel":{"type":"String"}
    }
  }
}
```

grounds or closures

Exterior: a) ../wtr/{gmlid}/{grounds or closures}/{gmlid}
b) ../wtr/{gmlid}/water



```
{
  "type":"Object",
  "properties":{
    "gmlid":{"type":"String"},
    "lod":{"type":"Number"},
    "waterLevel":{"type":"Number"},
    "geometry":{"type":"GeoJSON"},
    "generic":{"type":"Object"},
    "links":{"type":"Object",
     "properties":{
       "link":{"type":"String"},
       "rel":{"type":"String"}
      }
    }
  }
}
```

Available for "grounds" or "closures" resources

Available for "water" resource

ANNEX A: JSON-BASED SCHEMA OF SUB-RESOURCES

BIBLIOGRAPHY

BIBLIOGRAPHY

Athanasiou, K., Pispidikis, I., & Dimopoulou, E. (2018). Semantic-based Technologies for Interoperable BIM and GIS 3D Modelling, Storage and Retrieval. In *FIG Commission 3 Annual Meeting and Workshop 2018:* Conference, 3-6 December 2018. Italy: Naples.

Beil, C. & Kolbe, T. H. (2017). CityGML and the streets of New York - a proposal for detailed street space modelling, *ISPRS Ann. Photogramm. Remote Sens*. Spatial Inf. Sci., IV-4/W5, 9-16. Retrieved from: https://doi.org/10.5194/isprs-annals-IV-4-W5-9-2017

Butler, H., Daly, M., Doyle, A., Gillies, S., Hagen, S., & Schaub, T. (2016). *The geojson format*. Internet Engineering Task Force (IETF).

Cozzi, P, Lilley, S., & Getz, G. (2019). *OGC 3D Tiles Specification*. Version 1.0. (OGC Document Number 18-053r2).

Coors, V. (2018). *OGC Testbed-13: 3D Tiles and I3S Interoperability and Performance ER* (OGC Document Number 17-046).

Chaturvedi, K. & Kolbe, T H. (2017). *Future City Pilot 1 Engineering Report*. Retrieved from: http://docs.opengeospatial.org/per/16-098.html

Chaturvedi, K., Smyth, C. S., Gesquière, G., Kutzner, T., & Kolbe, T. H. (2017). Managing versions and history within semantic 3D city models for the next generation of CityGML. In *Advances in 3D Geoinformation,* 191-206. Springer, Cham.

Chaturvedi, K. & Kolbe, T. H. (2016). Integrating Dynamic Data and Sensors with Semantic 3D city models in the context of Smart Cities. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Science*s, IV-2/W1, 31–38. Retrieved from: https://doi.org/10.5194/isprs-annals-IV-2-W1-31-2016

Chaturvedi, K., Yao, Z., & Kolbe, T. H. (2015). Web based Exploration of and Interaction with Large and Deeply Structured Semantic 3D City Models using HTML5 and WebGL. In *Wissenschaftlich-Technische Jahrestagung der DGPF und Workshop on Laser Scanning Applications* (Vol. 3). Conference, 16-18 Marc 2015. Germany: Cologne.

Chaturvedi, K. (2014). *Web Based 3D Analysis and Visualization Using HTML5 and WebGL. University of Twente Faculty of Geo-Information and Earth Observation (ITC)*.

Chatzinikolaou, E., Pispidikis, I., & Dimopoulou, E. (2020). A Semantically enriched and Web-based 3D energy model visualization and retrieval from smart building implementation using CityGML and Dynamizer ADE, *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf.* Sci VI-4/W1-2020, 53–60. Retrieved from: https://doi.org/10.5194/isprs-annals-VI-4-W1-2020-53-2020

Chimezie, E. (2017). *REST versus GraphQL*. Retrieved from: https://blog.pusher.com/rest-versus-graphql/

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.

Falcor (n.d.). *A JavaScript library for efficient data fetching*. Retrieved from: https://netflix.github.io/falcor/

Fielding, R. T. & Taylor, R. N. (2000). *Architectural styles and the design of network-based software architectures*. Irvine: University of California.

Floros, G., Pispidikis, I., & Dimopoulou, E. (2017). Investigating integration capabilities between IFC and citygml LOD3 for 3D city modelling. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, *42*, 1.

Fowler, M. (2010). *Richardson Maturity Model: steps toward the glory of REST*. Retrieved from: http://martinfowler.com/articles/richardsonMaturityModel.html

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

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

Jo, J., Kim, Y., & Lee, S. (2014). Mindmetrics: Identifying users without their login IDs. In *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC).* International Conference 5-8 October 2020 (pp. 2121-2126). San Diego: IEEE.

Hartig, O. & Pérez, J. (2018). Semantics and complexity of GraphQL. In *Proceedings of the 2018 World Wide Web Conference*. (pp. 1155-1164). Conference, 23-27 April 2018: Lyon: University of Lyon.

Hagedorn, B., Thum, S., Reitz, T., Coors, C., & Gutbell R. (2017). *OGC® 3D Portrayal Service. Version 1.0* (OGC Document Number 15-001r4)

Hagedorn, B. (2010). *Web View Service Discussion Paper. Version 0.3.0*. OpenGIS Discussion Paper (OGC Document Number 09-166r2).

Helfer, J. (2016). *GraphQL vs. Falcor*. Retrieved from: https://blog.apollographql.com/graphql-vs-falcor-4f1e9cbf7504

Huang, X. (2002). *GeoAjent-based geospatial information service and application integration* (Phd dissertation). Beijing University.

Gaillard, J., Vienne, A., Baume, R., Pedrinis, F., Peytavie, A., & Gesquière, G. (2015). Urban data visualisation in a web browser. In *Proceedings of the 20th International Conference on 3D Web Technology* (pp. 81-88). ACM.

Gesquiere, G. & Manin, A. (2012). 3D Visualization of Urban Data Based on CityGML with WebGL. *International Journal of 3-D Information Modeling (IJ3DIM), 1(3)*, 1-15.

Gong, J. (1999). *Contemporary GIS theory and technology*. China: Wuhan University of Surveying and Mapping Science and Technology Press.

GraphQL is the better REST (n.d.). Retrieved from: https://www.howtographql.com/basics/1-graphql-is-the-better-rest/

Gröger, G., Kolbe, T., Nagel, C., & Hafele, K.-H. (2012). OGC City Geography Markup Language (CItyGML) Encoding Standard. Retrieved from: www.opengis.net/spec/citygml/2.0. (OGC Document Number 12-019)

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

Gutbell, R., Pandikow, L., Coors, V., & Kammeyer, Y. (2016). A framework for server side rendering using OGC's 3D portrayal service. In *Proceedings of the 21st International Conference on Web3D Technology* (pp. 137-146). ACM.

ISO, I. (2003). 19107: 2003 Geographic information-Spatial schema. *International Organization for Standardization*, 90.

ISO 14825, 2011. *Intelligent transport systems - Geographic Data Files (GDF) - GDF5.0*. International Standard, ISO.

Khronos Group (2019). *glTF Specification Webpage*. Retrieved from: https://www.khronos.org/gltf

Koukofikis, A., Coors, V., & Gutbell, R. (2018). Interoperable visualization of 3D City Models using OGC's Standard 3D Portrayal Service. ISPRS *Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, *4(4)*.

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

Kutzner, T., Chaturvedi, K., & Kolbe, T. H. (2020). CityGML 3.0: New Functions Open Up New Applications. *PFG–Journal of Photogrammetry, Remote Sensing and Geoinformation Scie*nce, 1-19.

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

Löwner, M.-O., Gröger, G., Benner, J., Biljecki, F., & Nagel, C. (2016). Proposal for a new lod and multi-representation concept for CityGML, *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci.,* IV-2/W1, 3-12. Retrieved from: https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/IV-2-W1/3/2016/isprs-annals-IV-2-W1-3-2016.pdf

Mao, B. & Ban, Y. (2011). Online Visualisation of a 3D City Model Using CityGML and X3DOM. *Cartographica*, *46(2)*, 109-114.

Mulligan, G. & Gračanin, D. (2009). 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.

OASIS. (2003). *xNAL Name and Address Standard*. Organization for the Advancement of Structured Information Standards. Retrieved from: http://xml.coverpages.org/xnal.html

OGC. (2007). *Summary of the OGC Web Services, Phase 4 (OWS-4) Interoperability Testbed* (OGC Document Number 07-037r4). Retrieved from: http://portal.opengeospatial.org/files/?artifact_id=21371

Ohori, K. A., Biljecki, F., Kumar, K., Ledoux, H., & Stoter, J. (2018). Modeling cities and landscapes in 3D with CityGML. In *Building Information Modeling* (pp. 199-215). Springer, Cham.

OpenGIS. (1999). *Consortium: OpenGIS Simple Features Specification for SQL, Revision 1.1*. OpenGIS Project Document, 99-049.

Peng, D., Li, C., & Huo, H. (2009). An extended usernametoken-based approach for rest-style web service security authentication. In *2009 2nd IEEE International Conference on Computer Science and Information Technology* (pp. 582-586). IEEE.

Pispidikis, I. & Dimopoulou, E. (2019). Conceptual model of CityGML RESTful Web Service, *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci.*, XLII-4/W15, 67–74. Retrieved from: https://doi.org/10.5194/isprs-archives-XLII-4-W15-67-2019

Pispidikis, I., Tsiliakou, E., Kitsakis, D., Athanasiou, K., Kalogianni, E., Labropoulos, T., & Dimopoulou, E. (2018). Combining methodological tools for the optimum 3D modelling of NTUA campus, *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci.*, IV-4/W6, 57-63. Retrieved from: https://doi.org/10.5194/isprs-annals-IV-4-W6-57-2018

Pispidikis, I. & Dimopoulou, E. (2018). CityGML RESTful Web Service: automatic retrieval of CityGML data based on their semantics. Principles, guidelines and bldg conceptual design, *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci.*, IV4/W6, 49-56. Retrieved from: https://doi.org/10.5194/isprs-annals-IV-4-W6-49- 2018

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 Ann. Photogramm. Remote Sens. Spatial Inf. Sci.*, IV-2/W1, 47-53. Retrieved from: https://doi.org/10.5194/isprs-annals-IV-2-W1-47-2016

Pispidikis, I. & Dimopoulou, E. (2015). Web development of spatial content management system through the use of free and open-source technologies. Case study in rural areas. *Journal of Geographic Information System*, *7(05)*, 527.

Portele, C. & Vretanos, P. (2018). *OGC Web Feature Service 3.0-Part 1: Core* (OGC Document Number 17-069r1). Retrieved from: http://docs.opengeospatial.org/DRAFTS/17-069r1.html

Portele, C., Vretanos, P., & Heazel Ch. (2019). OGC API-Features-Part 1: Core (OGC Document Number 17-069r3). Retrieved from: http://docs.opengeospatial.org/is/17-069r3/17-069r3.html

Portele, C. (2019). *OGC Testbed-14 Next Generation APIs: Complex Feature Handling Engineering Report* (OGC Document Number 18-021). Retrieved from: http://docs.opengeospatial.org/per/18-021.html

Prandi, F., De Amicis, R., Piffer, S., Soave, M., Cadzow, S., Gonzalez Boix, E., & D' Hont, E. (2013). Using CityGML to deploy Smart-City services for Urban Ecosystems. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Volume XL-4/W1, 87-92.

Prandi, F., Devigili, F., Soave, M., Di Staso, U., & De Amicis, R. (2015). 3D web visualization of huge CityGML models. *ISPRS International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *1*, 601-60.

Prieto, I., Izkara, L. J., & Del Hoyo, F. (2012). Efficient visualization of the geometric information of CityGML: application for the documentation of built heritage. *Computational Science and Its Applications–ICCSA* 2012, 529-544.

Quadt, U. & Kolbe, T. (2005). *Web 3D Service. Version 0.3.0* (OGC Document Number 05-019)

Reed, C. & Belayneh, T. (2017). *OGC Indexed 3d Scene Layer (I3S) and Scene Layer Package Format Specification. Version 1.0* (OGC Document Number 17-014r5)

Reitz, T. & Schubiger-Banz, S., (2014). The Esri 3D city information model. In *IOP Conference Series: Earth and Environmental Science*, *18(1)*. IOP Publishing.

Richardson, L. & Ruby, S. (2007). Web-services mit REST. O'Reilly Germany.

Rodriguez, A. (2008). *Restful web services: The basics*. IBM developer Works. Retrieved from: http://www.gregbulla.com/TechStuff/Docs/ws-restful-pdf.pdf

Schilling, A. & Kolbe, T. H. (2010). Draft for Candidate OpenGIS Web 3D Service Interface Standard. Version 0.4.0. *OpenGIS Discussion Paper* (OGC Document Number 09-104r1)

Schilling, A., Hagedorn, B., & Coors, V. (2012). OGC 3D Portrayal Interoperability Experiment Final Report. *OGC Engineering Report* (OGC Document Number 12-075)

Schilling, A., Bolling, J., & Nagel, C. (2016). Using glTF for streaming CityGML 3D city models. In *Proceedings of the 21st International Conference on Web3D Technology* (pp. 109-116). ACM.

Snowflake Software. (2016). *GO Publisher WFS Documentation. Introducing REST Services*. Retrieved from: https://wiki.snowflakesoftware.com/display/GPWFSDOC/Introducing REST Services

Somoza Alonso, F. (2015). *Development of a restful API: hateoas & driven API.*

Soon, K. H., & Khoo, V. H. S. (2017). CityGML modelling for Singapore 3D national mapping. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, *42*, 37.

Sudhakar, A. (2011). Techniques for securing REST. *CA Technology Exchange, 1*, 32-40.

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*, 1075-1082.

Tihomirovs, J. & Grabis, J. (2016). Comparison of soap and rest based web services using software evaluation metrics. *Information Technology and Management Science, 19(1)*, 92-97.

Vogel, M., Weber, S., & Zirpins, C. (2017). Experiences on migrating RESTful web services to GraphQL. In I*nternational Conference on Service-Oriented Computing* (pp. 283-295). Springer, Cham.

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

Wittern, E., Cha, A., & Laredo, J. A. (2018). Generating GraphQL-wrappers for REST (-like) APIs. In *International Conference on Web Engineering* (pp. 65-83). Springer, Cham.

Wilson, T. (2008). *OGC® KML. Version 2.2. 0* (OGC Document Number 07-147r2)

Whiteside, A. (2009). *Definition identifier URNs in OGC namespace. Version 1.3*. OpenGIS Best Practice document (OGC Document Number 07-092r3)

Whiteside, A. (2005). *GML 3.1.1 simple dictionary profile. Version 1.0.0* (OGC Document Number 05-099r2)

W3C. (2010). Same Origin Policy. Retrieved from: https://www.w3.org/Security/wiki/Same_Origin_Policy

Yao, Z., Nagel, C., Kunde, F., Hudra, G., Willkomm, P., Donaubauer, A., & 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)*, 1-26.

Zhu, W., Simons, A., Wursthorn, S., & Nichersu, A. (2016). Integration of CityGML and Air Quality Spatio-Temporal Data Series via OGC SOS. In *Proceedings of the Geospatial Sensor Webs Conference (GSW)*. Conference, 29-31 August 2016. (pp. 29-31). Muenster: 52north

LIST OF PUBLICATIONS

LIST OF PUBLICATIONS

PUBLICATIONS IN PEER-REVIEWED SCIENTIFIC JOURNALS

1. Floros, G., Tsiliakou E., Kitsakis, D., **Pispidikis I**., & Dimopoulou E. (2015). Investigating Semantic Functionality of 3D Geometry for Land Administration. In *Advances in 3D Geoinformation,* 247-264. Springer International Publishing.

2. Athanasiou, A., **Pispidikis, I**., & Dimopoulou, E. (2015). 3D Marine Administration System Based on LADM. In *Advances in 3D Geoinformation,* 385-407. Springer, Cham.

3. **Pispidikis, I**. & Dimopoulou, E. (2015). Web development of spatial content management system through the use of free and open-source technologies. Case study in rural areas. *Journal of Geographic Information System, 7(05)*, 527.

4. Floros, G., Solou, D., **Pispidikis, I.,** & Dimopoulou, E. (2016). A roadmap for generating semantically enriched building models according to CityGML model via two different methodologies.k, *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci., XLII-2/W2*, 23-32. Retrieved from: https://doi.org/10.5194/isprs-archives-XLII-2-W2-23-2016

5. **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 Ann. Photogramm. Remote Sens. Spatial Inf. Sci., IV-2/W1*, 47-53. Retrieved from: https://doi.org/10.5194/isprs-annals-IV-2-W1-47-2016

6. Floros, G., **Pispidikis, I.,** & Dimopoulou, E. (2017). Investigating integration capabilities between IFC and CityGML LoD4 for 3D City Modeling, *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci., XLII-4/W7*, 1-6. Retrieved from: https://doi.org/10.5194/isprs-archives-XLII-4-W7-1-2017

7. **Pispidikis, I.** & Dimopoulou, E. (2018). CityGML RESTful Web Service: automatic retrieval of CityGML data based on their semantics. Principles, guidelines and bldg

conceptual design, *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci., IV-4/W6*, 49-56. Retrieved from: https://doi.org/10.5194/isprs-annals-IV-4-W6-49-2018

8. **Pispidikis, I**., Tsiliakou, E., Kitsakis, D., Athanasiou, K., Kalogianni, E., Labropoulos, T., & Dimopoulou, E. (2018). Combining methodological tool for the optimum 3D modeling of NTUA Campus, *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci., IV-4/W6*, 57-63. Retrieved from: https://doi.org/10.5194/isprs-annals-IV-4-W6-57-2018

9. **Pispidikis, I.** & Dimopoulou, E. (2019). Conceptual model of CityGML RESTful Web Service, *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci., XLII-4/W15*, 67–74. Retrieved from: https://doi.org/10.5194/isprs-archives-XLII-4-W15-67-2019

10. Chatzinikolaou, E., **Pispidikis, I.**, & Dimopoulou, E. (2020). A Semantically enriched and Web-based 3D energy model visualization and retrieval from smart building implementation using CityGML and Dynamizer ADE, *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci., VI-4/W1-2020*, 53–60. Retrieved from: https://doi.org/10.5194/isprs-annals-VI-4-W1-2020-53-2020

PRESENTATIONS IN PEER-REVIEWED INTERNATIONAL CONFERENCES

1. **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. In *3D Geoinfo 2016*: Conference, 20-21 October 2016. Greece: Athens.

2. Floros, G., Solou, D., **Pispidikis**, **I.,** & Dimopoulou, E. (2016). A roadmap for generating semantically enriched CityGML model via two different methodologies. In *3D Geoinfo 2016*: Conference, 20-21 October 2016. Greece: Athens.

3. Floros, G., **Pispidikis**, **I.,** & Dimopoulou, E. (2017) Investigating integration capabilities between IFC and CityGML LoD4 for 3D City Modeling. In *3D Geoinfo 2017*: Conference, 26-27 October 2017. Australia: Melbourne.

4. **Pispidikis**, **I.** & Dimopoulou, E. (2018). CityGML RESTFul Web Service: Automatic retrieval of CityGML data based on their semantics. Principles, guidelines and BLDG conceptual design. In *3D Geoinfo 2018*: Conference, 1-2 October 2018. Netherlands: Delft.

5. **Pispidikis**, **I.,** Tsiliakou, E., Kitsakis, D., Athanasiou, K., Kalogianni, E., Labropoulos, T., & Dimopoulou, E. (2018). Combining methodological tool for the optimum 3D modeling of NTUA Campus. In *3D Geoinfo 2018*: Conference, 1-2 October 2018. Netherlands: Delft.

6. Athanasiou, K., **Pispidikis**, **I.,** & Dimopoulou, E. (2018). Semantic-based Technologies for Interoperable BIM and GIS 3D Modelling, Storage and Retrieval. In *FIG Commission 3 Annual Meeting and Workshop 2018*: Conference, 3-6 Decemger 2018. Italy: Naples.

7. Kitsakis, D., **Pispidikis, I.,** Athanasiou, K., Kalogianni, E. & Dimopoulou, E. (2019). Investigating the use of 3D Modelling and Geovisualisation for social housing. In *FIG Commission 3 Annual Meeting and Workshop 2019*: Conference, 23-28 Semtember 2019. Romania: Cluj-Napoca.

8. **Pispidikis, I.** & Dimopoulou, E. (2019). Conceptual model of CityGML RESTful Web Service. In *3D Geoinfo 2019*: Conference, 26-27 September 2019. Singapore.

9. **Pispidikis, I.,** Kitsakis, D., Kalogianni, E., Athanasiou, K., Lampropoulos, A., & Dimopoulou, E. (2020). 3D Modelling and Virtual Reality for the management of public buildings. In *FIG Working Week 2020. Smart surveyors for land and water management*: Conference, 10-14 May 2020. Netherlands: Amsterdam (submitted for publication).

OTHER PRESENTATIONS IN CONFERENCES

1. **Pispidikis, I**., & Dimopoulou, E. (2014). Web development of spatial content management system through the use of free and open-source technologies. Case study in rural areas. In *8th International Conference Hellas GIs*: Conference, 11-12 December 2014. Greece: Athens.

2. Koukoletsos T., **Pispidikis, I**., & Loisios, D., (2016). Design, development and integration of spatial analysis tools in WebGIS enviroment. In *24th National ArcGIS User Conference*: Conference, 19-20 May 2016. . Greece: Athens.

3. Koukoletsos T. & **Pispidikis**, I. (2017). Evaluation and Proposals for Parameterization of Geospatial Web Services. In *25th National ArcGIS User Conference*: Conference, **11-12 May 2017.** Greece: Athens.

4. Koukoletsos T. & **Pispidikis, I.** (2017). Latest technology WEBGIS Applications of Hellenic Military Geographical Service. In *25th National ArcGIS User Conference*: Conference, 11-12 May 2017. Greece: Athens.

5. **Pispidikis, I**., Tsiliakou, E., Kitsakis, D., Athanasiou, K., Kalogianni, E., Labropoulos, T. & Dimopoulou, E. (2017). Development of WebGIS platform for NTUA campus. In *25th National ArcGIS User Conference*: Conference, 11-12 May 2017. Greece: Athens.

6. **Pispidikis, I**., Tsiliakou, E., Kitsakis, D., Athanasiou, K., Kalogianni, E., Labropoulos, T., & Dimopoulou, E. (2017). Navigation in 3D virtual web environment – Implementation in SRSE NTUA. In *5th National Panhellenic Conference of Rural and Surveying Engineers*: Conference, 14-15 October 2017. Greece: Athens.

7. **Pispidikis, I**., Tsiliakou, E., Kitsakis, D., Athanasiou, K., Kalogianni, E., Labropoulos, T., & Dimopoulou, E. (2018). Development of a 3D web GIS application for NTUA

Campus. In *26th National ArcGIS User Conference*: Conference, 10-11 May 2018. Greece: Athens.

8. **Pispidikis, I.** (2018). New GeoIndex of Hellenic Military Geographical Service. In *26th National ArcGIS User Conference*: Conference, 10-11 May 2018. Greece: Athens.

9. Koukoletsos T., **Pispidikis, I.,** & Leader, D. (2018). Digital Interactive Map of Hellenic Military Geographical Service. In *26th National ArcGIS User Conference*: Conference, 10-11 May 2018. Greece: Athens.

CURRICULUM VITAE

PERSONAL INFORMATION

Surname:	Pispidikis
Name:	Ioannis
Father's name:	Stauros
Mother's name:	Dimitra
Date of birth	18 February 1984
Birth place:	Athens
Family status:	Married (Gkagklou Antonia)
Address	Mithridatou 36-38, 11632, Athens
Phone	+302111845624 (Home) +302108206686 (Work) +306951762683 (mobile)
e-mail	pispidikisj@yahoo.gr jpispidikis@gmail.com

EDUCATION AND STUDIES

2016-2020 : Ph.D Candidate, National Technical University of Athens. Geomatics.

2016 : Master's Degree (M.Sc), National Technical University of Athens. Geomatics, Grade: **9.33/10 (Excellent)**

2014 : Diploma, National Technical University of Athens, School of Rural and Surveying Engineering. Grade: **9.17/10 (Excellent)**

2010 : Batchelor's Degree, Topography School in Hellenic Military Geographical Service (HMGS), Surveying Engineering.

2005 : Batchelor's Degree, Hellenic Military Academy, Military Operational Art and Science/Studies

2001 : General High School. Grade: 18/20

SCIENTIFIC TRAINING

07/09 – 11/09/20 : 15th 3D Geoinfo Conference, United Kingdom: London

25/09 – 27/09/19 : 14th 3D Geoinfo Conference, Singapore

01/10 – 02/10/18 : 13th 3D Geoinfo Conference, Netherland: Delft

20/10 – 21/10/16 : 11th 3D Geoinfo Conference, Greece: Athens

18/10 – 20/10/16 : 5th International Workshop on 3D Cadastres, Greece: Athens

25/05 – 27/05/15 : Training Program «The Common Assessment Framework as a Tool for Total Quality Management. Training Institute

02/03 – 12/06/15 : Education for aerial photography. Army Aviation School

11 – 12/12/14 : 8th National Conference Hellas GIs. National Technical University of Athens.

9 – 10/10/14 : Seminar: «Introduction to Geoprocessing Scripts Using Python». Marathon Data Systems

25 – 26/9/14 : Seminar: «ArcGIS for Server-Sharing GIS Content on the Web» version 10.x. Marathon Data Systems

22, 23 & 24/9/14 : Seminar: «Introduction to Geographic Information System (GIS) » ArcGIS Extensions (3D Analyst-Spatial Analyst) version 10.x.
Marathon Data Systems

15, 16 & 17/9/14 : Seminar: «Introduction to Geographic Information System (GIS)» ArcGIS II version 10.x. Marathon Data Systems

8, 9 & 10/9/14 : Seminar: «Introduction to Geographic Information System (GIS)»ArcGIS I version 10.x. Marathon Data Systems

PROFESSIONAL EXPERIENCE

2014-Today : Director of Geodatabases Subdivision. Hellenic Military Geographical Service (HMGS). Athens, Greece.

- Supervision of day to day operations
- Military geospatial application development (Intranet WebGIS application, Android app)
- Data collection, manipulation and validation
- Installation and maintenance of all geodatabases for development purposes

2008-Today : Officer of the Greek Army in the Hellenic Military Geographical Service and Hellenic Military Topographic Service.

2005-Today : Officer in Greek Army

Academic Years: 2016/2017 & 2017/2018 : Member of the teaching team of the course: "Cadastral and Land Policy Systems", 8th semester, School of Rural and Surveying Engineering, National Technical University of Athens

20-21/10/2016 : Member of the Organizing Committee: "11th 3D Geoinfo Conference", Greece: Athens

18-20/10/2016 : Member of the Organizing Committee: "5th International Workshop on 3D Cadastres", Greece: Athens

2009-2015 : Head of department in the execution of several topographic surveys in Greece:

- topographic surveys
- cadastral surveys
- expropriations
- delimitation
- property surveying

2005-2008 : Officer of the Greek Army, Infantry Specialty, as Trainer and Staff Commander

HONORS & AWARDS

2/2019 : Ethic award from chief of Hellenic Army. Development and upgrading of several software of HMGS. As a result, the geospatial support of the Hellenic Army is improved and the work of the Hellenic Military Geographical Service is also promoted

5/2018 : Thomaidis award for publishing the paper with title: "Investigating integration capabilities between IFC and CityGML LoD3 for 3D City Modeling"

10/2017-10/2020 : Scholarship from ONASSIS FOUNDATION for Ph.D research

9/2017 : Award from Technical Chamber of Greece. Second highest grade for the School of Rural and Surveying Engineering among 2014 graduate

2017 : Ethic award from chief of Hellenic Army. Academic Excellence in master studies

5/2017 Thomaidis award for publishing the paper with title: "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"

2015/2016 : Scholarship from Zwh Soutsou legacy for Ph.D research.

9/2015 : Thomaidis award. Best Undergraduate student in 2013-2014 among all students of the school of Rural and Surveying Engineering

2015 : Ethic award from chief of Hellenic Army. Academic Excellence in bachelor studies

7/2015 : Award of Academic Excellence. Second highest grade for the School of Rural and Surveying Engineering among 2014 graduate

5/2015 : Thomaidis award for publishing the paper with title: "Web Development of Spatial Content Management System through the Use of Free and Open-Source Technologies. Case Study in Rural Areas"

2/2015 : Award of Academic Excellent. LIMMAT STIFTUNG-Memorandum of Agreement 29502/14

2000 : 5th Position in Nationwide Contest «LYSIAS»

THESES-DISSERTATIONS

I. Pispidikis, «Optimization of automated retrieval of semantic 3D city data», **PhD Thesis, Geomatics, National Technical University of Athens, 2020**

I. Pispidikis, «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», **Master Thesis, Geomatics, National Technical University of Athens, 2016**

I. Pispidikis, «Web Development of Spatial Content Management System through the Use of Free and Open-Source Technologies. Case Study in Rural Areas», **Bachelor Thesis, School of Rural and Surveying Engineering, National Technical University of Athens, 2014**

PROGRAMMING SKILLS

- Very Good Programming Knowledge: (Visual Basic, C++, C#, C)
- High level Front-end Web Development Skills (JavaScript, HTML, CSS etc.)
- High Level Back-end Web Development Skills (Web Services, Node, REST API, PHP, Java, python, MySQL, SQLite, PostgreSQL/PostGIS, ArcSDE, ArcGIS Server, GeoServer, MapServer etc.)
- High Level knowledge of GIS and CAD application such as AutoCAD, ArcGIS and QGIS
- High Level Knowledge and experience of developing WebGIS and Cross-Platform GIS applications