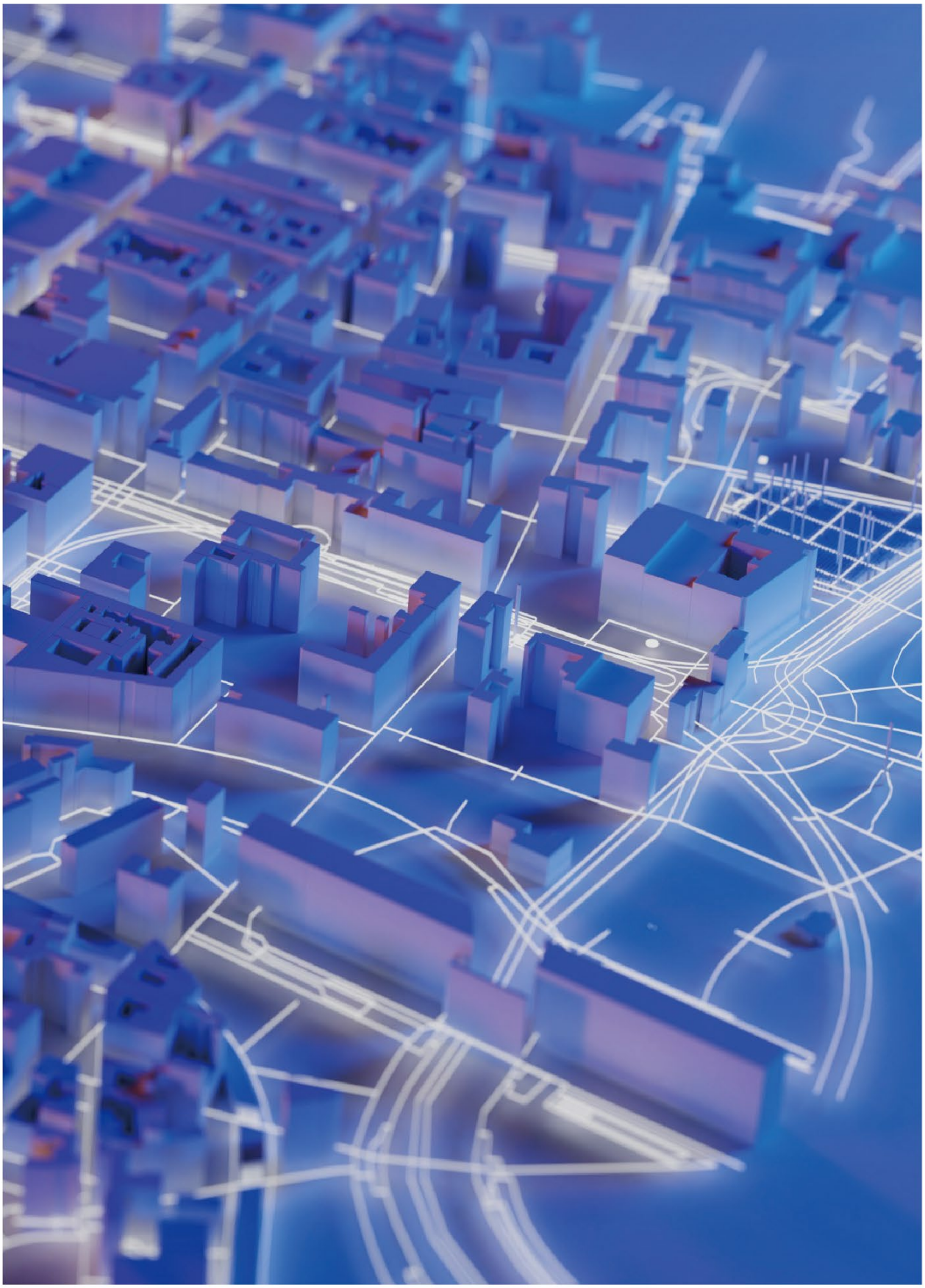
PLATEAU by MLIT

PLATEAU Technical Report

3D都市モデル活用のための技術資料



# CityGML 3.0 技術仕様調査レポート (2022年度)

series No. 57

Technical Report on OGC City Markup Language (CityGML) 3.0

# 目次

## はじめに

地理空間情報に関する国際標準化団体である Open Geospatial Consortium（OGC）は、3D 都市モデルを蓄積・交換するための概念モデルと符号化仕様（フォーマット）の国際標準として、「City Geography Markup Language（CityGML）」を策定した。

CityGML は、都市全体をモデル化することを目的とし、建築物、橋梁、トンネル、道路、地形のような都市を構成する様々なオブジェクトを定義している。また、LOD（Levels Of Detail）と呼ばれる一つのオブジェクトを複数の詳細度で表現可能なマルチスケールな仕組みを有し、目的に応じて異なる詳細度で記述された空間属性（幾何オブジェクト）を一つの 3D 都市モデルの中で統合的に管理できる。さらに、応用分野に固有の地物やその属性を追加する ADE（Application Domain Extensions）の仕組みにより、3D 都市モデルの利用目的や都市の規模・環境に応じて、必要な情報を拡張できる。こうした CityGML の特徴を踏まえ、国土交通省が主導する 3D 都市モデルのオープンデータ化プロジェクト“PLATEAU”では、3D 都市モデルの概念モデル及び符号化仕様として CityGML を採用している。3D 都市モデルは、従来は個別に存在する統計値でしかなかった都市計画基礎調査などの情報を、3 次元の空間属性とともに「属性」として統合したデータとして整備されている[1]。

CityGML は 2008 年 8 月に第 1.0 版（CityGML 1.0）が発行され、2012 年 4 月に第 2.0 版（CityGML 2.0）が発行された。さらに、2021 年 9 月に、「OGC City Geography Markup Language (CityGML) Part 1: Conceptual Model Standard」[2]として、CityGML 第 3.0 版（CityGML 3.0）の概念モデルが発行された。また、その符号化仕様である「OGC City Geography Markup Language (CityGML) 3.0 Part 2: GML Encoding Standard」[3]が 2023 年 6 月末に発行された。

今後、CityGML 2.0 から CityGML 3.0 へ移行していくことが想定されるが、CityGML 3.0 の解説資料はまだ少なく移行する利点や移行に向けた課題は明らかではない。そこで、本レポートでは、OGC が公開する CityGML 3.0 の標準文書、CityGML 3.0 の開発過程において発表された論文及び技術レポート、また、CityGML 3.0 を開発する OGC 内の仕様策定ワーキンググループ（Standards Working Group：SWG）である CityGML SWG が管理する技術資料のポータルサイト[4]から公開されている関連資料を整理し、CityGML 3.0 の改定内容を解説する。資料の整理にあたり、CityGML SWG メンバとの意見交換を参考とした。

第 1 章では、CityGML 3.0 を概説する。第 2 章では、CityGML 3.0 の概念モデルを解説する。第 3 章では、CityGML 3.0 の GML による符号化仕様及びその実装方法を解説する。

# 1. CityGML 3.0 概要

## 1.1. CityGML とは

CityGML は、3D 都市モデルを蓄積・交換するための概念モデルと符号化仕様（フォーマット）を定める標準である。

図 1-1 に CityGML が定める範囲を示す。概念モデルとは、3D 都市モデルに含まれるべき地物、地物の属性及び地物間の関係（データ構造）を示すものであり、UML クラス図を用いて記述される。符号化仕様とは、概念モデルに従ったデータを符号化するための仕様である。CityGML では地理空間情報に特化した XML ベースの Geography Markup Language（GML）が採用され、この GML で 3D 都市モデルを符号化するための XML Schema を定めている。XML Schema は、概念モデルに含まれる地物、地物の属性及び地物間の関係に対応する XML のタグやタグの出現順序及び出現回数を指定し、この XML Schema に従って作成されたデータが 3D 都市モデルのデータとなる。

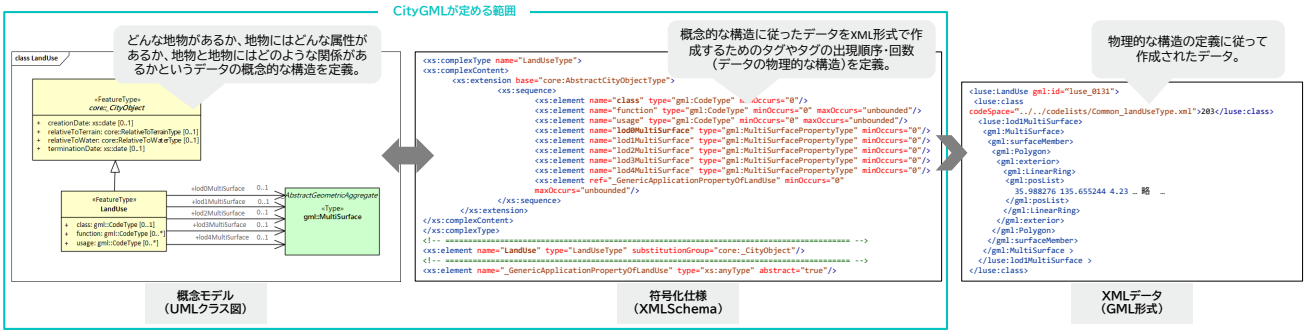



**図 1-1 CityGML が定める範囲**

## 1.2. CityGML 3.0 の改定のポイント

CityGML 3.0 では、IoT やスマートシティといった分野で 3D 都市モデルをより高度な分析やシミュレーションに活用できるようにすること、そして 3D 都市モデルに関連する他の標準との相互流通性を向上することを目的とする改定が行われた。他の標準とは、BIM の標準である Industry Foundation Classes (IFC)、屋内空間やナビゲーションのための標準である IndoorGML、欧州空間情報基盤（INSPIRE）、地籍など土地管理のための標準である Land Administration Domain Model (LADM)及びセマンティックウェブにおけるリソースに関する情報の表現方法を定める Resource Description Framework (RDF)である。

CityGML 2.0 から CityGML 3.0 への主な改定のポイントを以下に示す。

(1) 概念モデルと符号化仕様の分離

CityGML 1.0 及び CityGML 2.0 は、概念モデルと符号化仕様が一体的な国際標準として発行されたが、CityGML 3.0 では、概念モデルと符号化仕様を分離し、概念モデルは Part1、GML による符号化仕様は Part2 と分けて発行されることとなった。概念モデルと符号化仕様の分離は、地理空間情報に関する国際規格を定める ISO/TC211 により策定された ISO 19109 応用スキーマのための規則及び ISO 19118 符号化に準拠した考え方である。概念モデルと符号化仕様を分離することにより、今後、GML 以外の符号化仕様を Part3 以降で標準化する可能性がある。詳細は 3.1 で説明する。

(2) LOD の見直し

CityGML 2.0 では、LOD（Levels Of Detail：詳細度）を、LOD0 から LOD3 までを屋外や外部のモデル表現、LOD4 を屋内や内部のモデル表現とする 5 段階に区分していた[5]。CityGML 3.0 では、LOD0 から LOD3 の 4 段階となり、LOD4 を廃止した。LOD0 から LOD3 の各段階を、屋外・外部だけではなく屋内・内部にも適用することになり、これにより屋内・内部の空間を LOD0 から LOD3 の 4 段階の詳細度で表現することが可能となった。詳細は 2.2.2 で説明する。

(3)　空間属性の統合

CityGML 2.0 では、建築物（*Building*）、土地利用（*LandUse*）などの都市オブジェクトごとに空間属性（*Solid* や *MultiSurface* などの幾何オブジェクトへの参照）を定義していた。これが、CityGML 3.0 では、都市オブジェクトごとの空間属性の定義はなくなり、上位概念となる地物にまとめて定義した。各都市オブジェクトは、上位概念に定義された空間属性を、継承によりもつことができる。CityGML 2.0 では、都市オブジェクト毎に空間属性を定義していたため、例えば、建築物の付属物（*BuildingInstallation*）は LOD2 以上、窓（*Window*）は LOD3 以上というように、LOD によって表現可能な都市オブジェクトが異なっていた。CityGML 3.0 では、上位概念に定義された LOD0 から LOD3 までの空間属性を全ての都市オブジェクトが継承するため、各 LOD で全ての都市オブジェクトを表現可能となる。詳細は 2.2.3 で説明する。

(4)　新たなモジュールの追加

CityGML 3.0 では、新たに Construction、Dynamizer、Versioning 及び PointCloud の 4 モジュールを追加した。Construction モジュールは、建築物、橋梁及びトンネル以外の構造物を定義する。Dynamizer モジュールは、センサーデータなど動的な時系列データを定義する。Versioning モジュールは 3D 都市モデルやそれに含まれる都市オブジェクトの新旧を関連付ける。PointCloud モジュールは点群を定義する。各モジュールの詳細は、2.6 から 2.9 で説明する。

(5)　新たな概念の追加

CityGML 3.0 では、新たに「空間（Space）」と「境界（Space Boundary）」という概念を追加した。建築物や道路など全ての都市オブジェクトは、空間又は境界のいずれかに区分される。詳細は 2.2.1 で説明する。また、地物がいつ生じていつ消滅したかという地物の時間的な性質（時間属性）を追加した。詳細は 2.2.4 で説明する。

## 1.3.　CityGML 3.0 のモジュール

CityGML は概念毎にモジュール（パッケージ）が作成され、CityGML 3.0 は全部で 17 のモジュールから構成される（図 1-2）。モジュールは、その性質により以下の三つに区分できる。

(1)　建築物（Building）や土地利用（LandUse）、交通（Transportation）、植生（Vegetation）等の 3D 都市モデルを構成する地物を定義する、主題毎に作成されたモジュール（「主題モジュール」と呼ぶ）

(2)　テクスチャやセンサーデータのように、全ての主題モジュールが利用可能な概念を定義するモジュール

(3)　全てのモジュールに継承される基本的な概念と空間属性を定義するモジュール

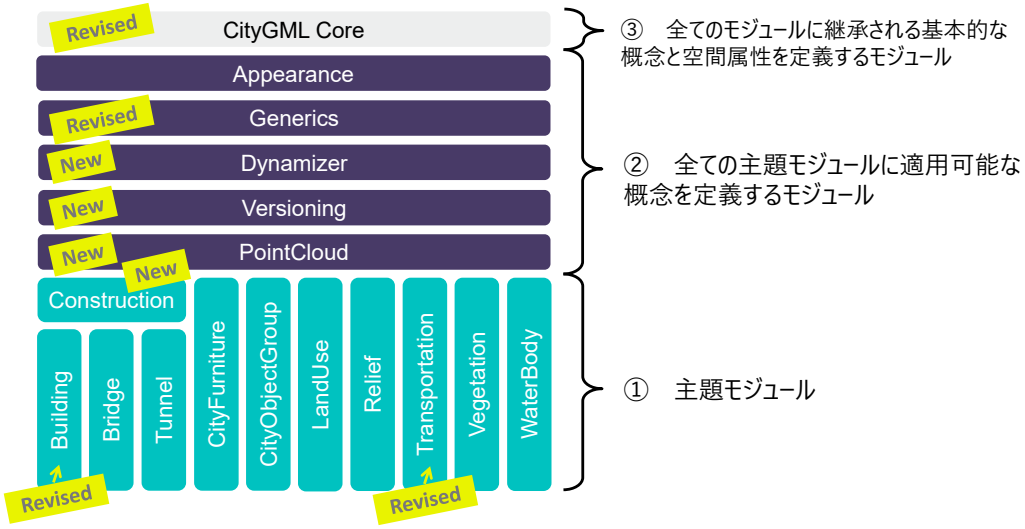
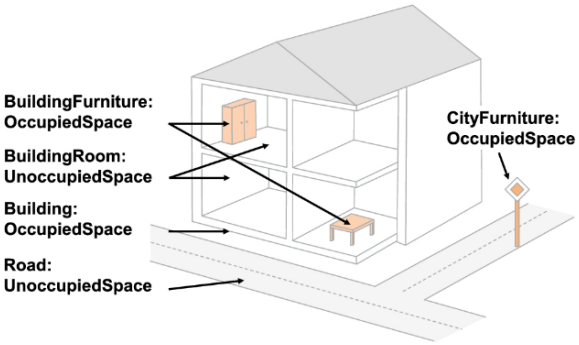
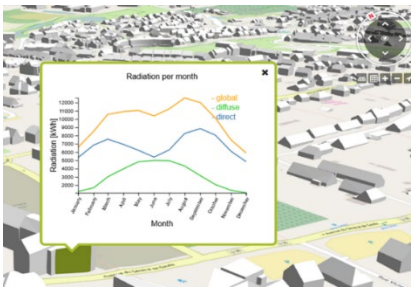
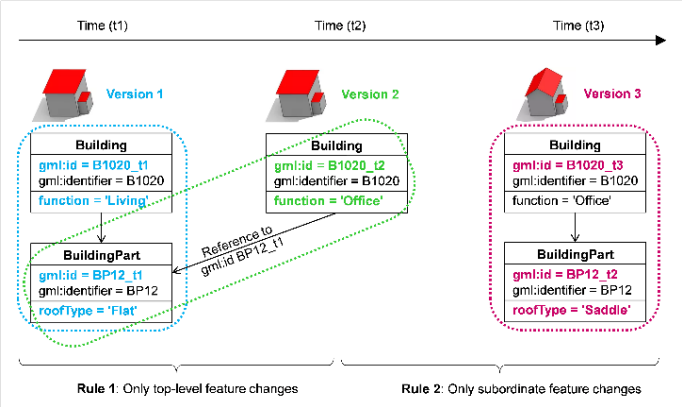
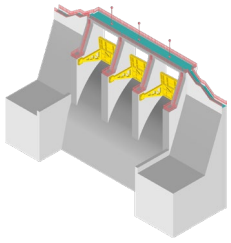


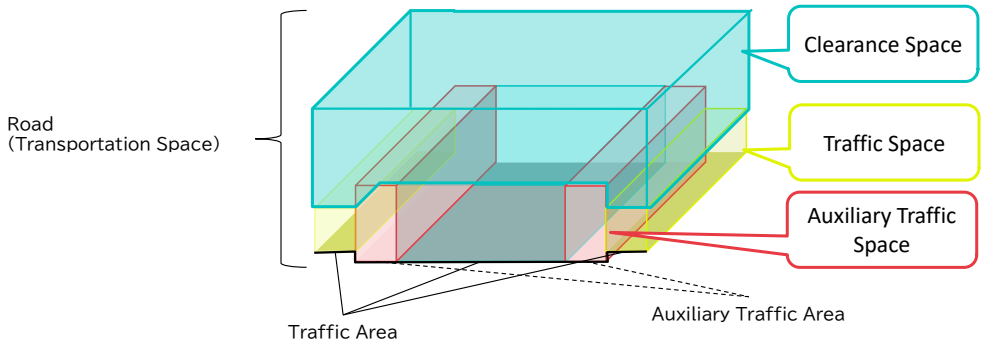

**図 1-2　CityGML 3.0 のモジュール構成（出典[4]一部補足）**

図 1-2 において“Revised”と示されたモジュールは、CityGML 2.0 から改定されたモジュールである。また、“New”と示されたモジュールは、CityGML 3.0 で新たに追加されたモジュールである。各モジュールの概要と今回の改定における区分（追加、改定又は変更無し）を表 1-1 に示す。改定がないモジュールもあるが、Core モジュールが改定されているため、Core モジュールを継承する全てのモジュールに CityGML 3.0 の改定による影響が生じる。

**表 1-1　CityGML 3.0 のモジュール**

| モジュール | 概要及び主な改定内容 | 区分 |
|---|---|---|
| Core | 全てのモジュールが継承する基本的な概念と空間属性を定義する。<br>これまで各モジュールで定義されていた幾何オブジェクトへの参照が、Core モジュールに統合され、全てのモジュールで共通となった。併せて、LOD の概念が変更された。また、新たに *Space*（空間）と *SpaceBoundary*（境界）という概念が定義され、全ての地物は *Space* 又は *SpaceBoundary* のいずれかを継承する。Space には、物により占有される空間（*OccupiedSpace*）と物により占有されていない空間（*UnoccupiedSpace*）に区分できる（下図）。さらに、時間概念をもつ地物 *AbstractFeatureWithLifeSpan* が定義され、地物の生起を表現可能となった（詳細は 2.2 で説明）。<br>  <br><br>出典[2]　©OGC | 改定 |
| Appearance | 地物の外観（テクスチャ、表面色）を定義する。 | ― |
| Generics | 汎用都市オブジェクト及び汎用属性を定義する。<br>CityGML で定義されていない地物や属性を追加する際に使用できる。CityGML 3.0 では、Core モジュールに定義された空間や境界などの概念ごとに汎用都市オブジェクトを定義する。また、汎用属性として、コード型を追加した。（詳細は 2.3 で説明） | 改定 |
| Dynamizer | 地物とセンサーデータを連携できる。例えば、屋根面の地物にセンサーから取得した日射量を連携することなどが可能となる。（詳細は 2.7 で説明）<br>  <br><br>出典[4]　©OGC | 追加 |

| モジュール | 概要及び主な改定内容 | 区分 |
|---|---|---|
| Versioning | 地物やデータの版管理の仕組みを定義する。<br>二つのデータセットの新旧や、改築前の建築物と改築後の建築物というように、データセット内の地物をバージョン管理し対応付けることができる。（詳細は 2.8 で説明）<br>  <br>出典[4]　©OGC | 追加 |
| PointCloud | 点群データを定義する。<br>地物の形状表現として点群データを使用できる。例えば、建築物に対して、その建築物に関係する点群データを関連付けることができる。（詳細は 2.9 で説明）<br>  <br>出典[4]　©OGC | 追加 |
| Construction | 構造物を定義する。<br>建築物、橋梁、トンネルの上位概念となる。構造物に共通する境界面及び開口部は、Construction モジュールに定義された。また、*OtherConstruction* が追加され、ダム（右図）のように、橋梁、トンネル及び建築物以外の構造物を記述するために使用できる。（詳細は 2.6 で説明）<br>  | 追加 |
| Building | 建築物及びこれを構成する地物を定義する。<br>屋内の表現として、新たに *Storey*（階）及び *BuildingUnit*（区画）を追加した。また、*BuildingConstructiveElement* の追加により、建築物を構成する部材を表現可能となった。（詳細は 2.4 で説明） | 改定 |
| Bridge | 橋梁及びこれを構成する地物を定義する。 | — |
| Tunnel | トンネル及びこれを構成する地物を定義する。 | — |
| CityFurniture | 都市設備（信号機や照明施設などの小規模な設備）を定義する。 | — |
| CityObjectGroup | 都市オブジェクトの集まりを定義する。 | — |
| LandUse | 土地利用を定義する。 | — |

| モジュール | 概要及び主な改定内容 | 区分 |
|---|---|---|
| Relief | 地形を定義する。 | ― |
| Transportation | 交通オブジェクト（道路、鉄道、徒歩道、航路及び広場）を定義する。<br>交通オブジェクトは、空間として表現可能となった（下図）。これによりロボットが通行可能な空間やドローンが飛行可能な空間などを記述できる。また、車道毎、車線毎というように情報の粒度を選択できる。さらに、区間と交差点に分けて記述できるようになった。<br>  | 改定 |
| Vegetation | 植生を定義する。 | ― |
| WaterBody | 水部を定義する。 | ― |

―は変更無し

## 2. CityGML 3.0 概念モデル

### 2.1. はじめに

CityGML 3.0 Part 1 では、新たに追加されたモジュール（Dynamizer、Versioning、PointCloud、Construction）の概念モデルが追加された。また、Core、Generic、Building 及び Transportation の 4 モジュールは概念モデルが改定された。

本項では、改定及び追加されたモジュールの概念モデルを解説する。

### 2.2. Core モジュール

Core モジュールは、CityGML 3.0 に定義された全ての地物に共通となる概念を定義するモジュールである。以下に示す Core モジュールの改定内容は、*Building* や *Bridge* をはじめとする全ての地物に適用される。

#### 2.2.1. 空間及び境界の追加

CityGML 3.0 では、新たな概念として「空間（*Space*）」とその「境界（*SpaceBoundary*）」を導入した。建築物や道路など、主題モジュールに定義される全ての地物は、空間又は境界のいずれかに区分される。都市オブジェクト（*AbstractCityObject*）は、主題モジュールで定義される全ての地物の最上位概念である。これを継承し、空間を示す地物である *AbstractSpace* と、空間の境界となる地物である *AbstractSpaceBoundary* が定義された（図 2-1）。また、*AbstractSpace* を継承し、論理的な空間（*AbstractLogicalSpace*）と物理的な空間（*AbstractPhysicalSpace*）が定義される。さらに、物理的な空間を継承し、占有された空間（*AbstractOccupiedSpace*）及び占有されていない空間（*AbstractUnoccupiedSpace*）が定義される。また、空間の境界である *AbstractSpaceBoundary* を継承する *AbstractThematicSurface* は、主題モジュールに定義される面的な地物の最上位概念であり、占有空間又は非占有区間として表現される地物の境界面（例：壁面、道路面）として使用される。

CityGML 3.0 の主題モジュールに定義される全ての地物は、*AbstractLogicalSpace*、*AbstractOccupiedSpace*、

*AbstractUnoccupiedSpace* 又は *AbstractSpaceBoundary* のいずれかを継承する。つまり、建築物、トンネル、道路、土地利用といった全ての都市オブジェクトは、「論理的な空間」「占用された空間」「占用されていない空間」又は「主題的な面」のいずれかに区分される。さらに、*AbstractThematicSurface* を継承し、仮想的な面である *ClosureSurface* が定義されている。*ClosureSurface* は、壁面のような物理的な地物が存在しない場所で空間となる地物を区切るために使用する。CityGML 2.0 では、各モジュールに *ClosureSurface* が定義されていたが、CityGML ではこれを Core モジュールに定義し、共通の地物として利用できるようにした。

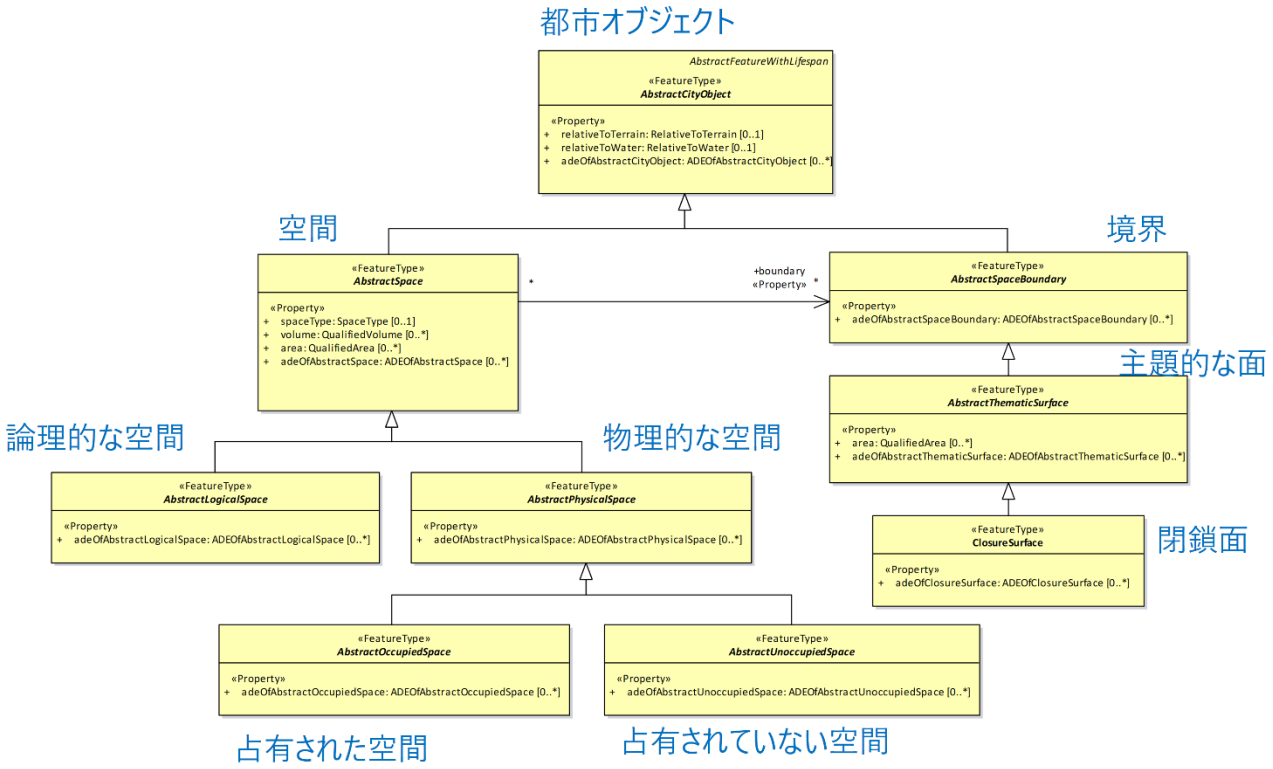



**図 2-1　空間（AbstractSpace）とその境界（AbstractSpaceBoundary）の概念モデル（出典[2]一部補足）**

都市オブジェクトは、CityGML 2.0 と同様に、属性として地表面との相対的な位置関係を表す属性（*relativeToTerrain*）と水面との相対的な位置関係を示す属性（*relativeToWater*）をもつ。また、Application Domain Extension（ADE）による拡張の仕組み（“hook”と呼ばれる）を使用するための属性 *adeOfAbstractCityObject* が定義される。CityGML 2.0 では、ADE の仕組みは UML クラス図には記載されず、XML Schema にのみ記載されていた。CityGML 3.0 では、概念モデルと符号化仕様を 1 対 1 で対応させるため、ADE による拡張の仕組みも地物の属性として UML クラス図に記載されている。CityGML 3.0 に定義される全ての地物は、ADE による拡張の仕組みが付与され、UML クラス図では、地物の属性 *adeOfXXX*（XXX には、地物の名称が入る）として表現される。

都市オブジェクトを継承する *AbstractSpace* にはその空間がオープンか閉じられているのかといった区分を示す属性 *spaceType* のほか、空間の面積（*area*）、体積（*volume*）及び ADE のための属性（*adeOfAbstractSpace*）をもつ。

*AbstractLogicalSpace* は、CityGML3.0 で追加された新たな地物である。CityGML 2.0 では、目に見える物理的な地物しか定義されていなかった。*AbstractLogicalSpace* は、行政区域や都市計画区域といった目に見えない仮想的な空間を表現する概念である。ただし、*AbstractLogicalSpace* は抽象クラスであり、インスタンス化（データ化）できない。そのため、2.3 で説明する汎用的な論理空間オブジェクトである *GenericLogicalSpace* を使用するか、*AbstractLogicalSpace* を継承し、インスタンス化が可能な具象クラスを ADE に定義する必要がある。

*AbstractPhysicalSpace* は物理的な空間である。物理的な空間は、*AbstractOccupiedSpace* 又は *AbstractUnoccupiedSpace* のいずれかに区分される。例えば、建築物は空間を「もの」（建築物）が占有しているため、*AbstractOccupiedSpace* を継承する。道路空間や部屋のように、「もの」が無い空間は、*AbstractUnoccupiedSpace* を継承する。一方、道路上の都市設備や部屋の中にある家具は、建築物と同様に空間を占有しているため、*AbstractOccupiedSpace* を継承する（図 2-2）。このような「占有されている空間」や「占有されていない空間」といった概念は、自動運転やロボットの走行等の障害物判定に利用することを目的として、CityGML 3.0 において新たに採用された概念である。

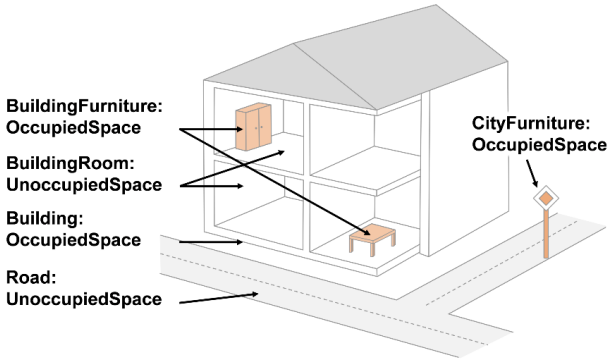



**図 2-2　占有空間（Occupied Space）と非占有空間（Unoccupied Space）（出典[2] ©OGC）**

このような空間の概念は、地籍の 3 次元表現に活用することが期待されている[9]。特に欧州では大都市を中心に土地が不足しており、年々建築物の構造が複雑化している。これに伴い、建築物には複数の用途や所有者が存在し、用途や所有者に応じた、異なる権利や義務が与えられている。2 次元の GIS データでは表現できない 3 次元での地籍管理が、地物を空間として捉えることで実現できる。

表 2-1 は、各主題モジュールに定義された地物（ただし、インスタンス化可能な具象クラスのみ）が、「論理的な空間」「占用された空間」「占用されていない空間」又は「主題的な面」のいずれに区分されるかを示したものである。また、表 2-1 において太字で記載している地物は、CityGML 3.0 で新たに追加された地物である。

**表 2-1　空間及び境界の区分**

| SuperClass（上位型） | TopLevelFeatureType | FeatureType |
|---|---|---|
| *AbstractLogicalSpace* | *CityObjectGroup* 、 ***GenericLogicalSpace*** | — |
| *AbstractOccupiedSpace* | *CityFurniture* 、 *SolitaryVegetationObject* 、 *PlantCover* 、 *WaterBody* 、 ***OtherConstruction*** 、 *Bridge* 、 *Building* 、 *Tunnel* 、 ***GenericOccupiedSpace*** | ***Window*** 、 ***Door*** 、 *BridgePart* 、 *BridgeConstructiveElement* 、 *BridgeRoom* 、 *BridgeInstallation* 、 *BridgeFurniture* 、 *BuildingPart* 、 ***BuildingConstructiveElement*** 、 *BuildingRoom* 、 *BuildingInstallation* 、 *BuildingFurniture* 、 ***BuildingUnit*** 、 ***Storey*** 、 *TunnelPart* 、 ***TunnelConstructiveElement*** 、 *HollowSpace* 、 *TunnelInstallation* 、 *TunnelFurniture* |

| | | |
|---|---|---|
| *AbstractUnoccupiedSpace* | *Road*、*Track*、***Waterway***、*Railway*、*Square*、*GnericUnoccupiedSpace* | ***Section***、***Intersection***、***ClearanceSpace***、***TrafficSpace***、***AuxiliaryTrafficSpace***、***Hole*** |
| *AbstractSpaceBoundary* | *LandUse*、*ReliefFeature*、***GenericThematicSurface*** | *TINRelief*、*MassPointRelied*、*BreaklineRelief*、*RasterRelief*、*TrafficArea*、*AuxiliaryTrafficArea*、***HoleSurface***、***Marking***、*WaterSurface*、*WaterGroundSurface*、*DoorSurface*、*WindowSurface*、*ClosureSurafce*、*WallSurface*、*RoofSurface*、*GroundSurface*、*FloorSurface*、*CeilingSurface*、*OuterCeilingSurface*、*OuterFloorSurface*、*InteriorWallSurface* |

なお、CityGML 3.0 では地物を示すクラスのステレオタイプとして、<<TopLevelFeatureType>>と<<FeatureType>>を使用している。<<TopLevelFeatureType>>は、3D 都市モデルにおいて、主要な概念を示す地物である。<<FeatureType>>は、<<TopLevelFeatureType>>の部品となる地物に付与するステレオタイプである。<<FeatureType>>は、これ単体で 3D 都市モデルに含まれることはなく、常に<<TopLevelFeatureType>>の部品として存在する。

### 2.2.2. LOD の見直し

CityGML 3.0 では、LOD の概念が改定された。CityGML 2.0 では、LOD を 0 から 4 までの 5 段階に分け、LOD0 は 2 次元と 3 次元の連携、LOD1 は箱モデル、LOD2 はそれを詳細化し、LOD3 は外部（屋外）の最も詳細なモデル、LOD4 は内部（屋内）のモデルとして位置づけた。これに対し、CityGML 3.0 では、LOD を 0 から 3 までの 4 段階に区分する。LOD4 は廃止となった。しかしこれは屋内が対象外となったのではなく、屋内も屋外と同様に LOD0 から 3 の詳細度に分けて表現できるようになった。CityGML 2.0 では、屋内の表現は LOD4 として BIM モデルを活用した最も詳細な定義のみであったが、CityGML 3.0 では、屋内の部屋も LOD0 では点、線又は面、LOD1 では立体として表現できる。例えば、LOD0 は簡易なフロアマップ、LOD1 は部屋の箱モデル、LOD2 では部屋の壁面や天井の形状を記述する、というように目的に応じて詳細度を変えて表現できる（図 2-3）。

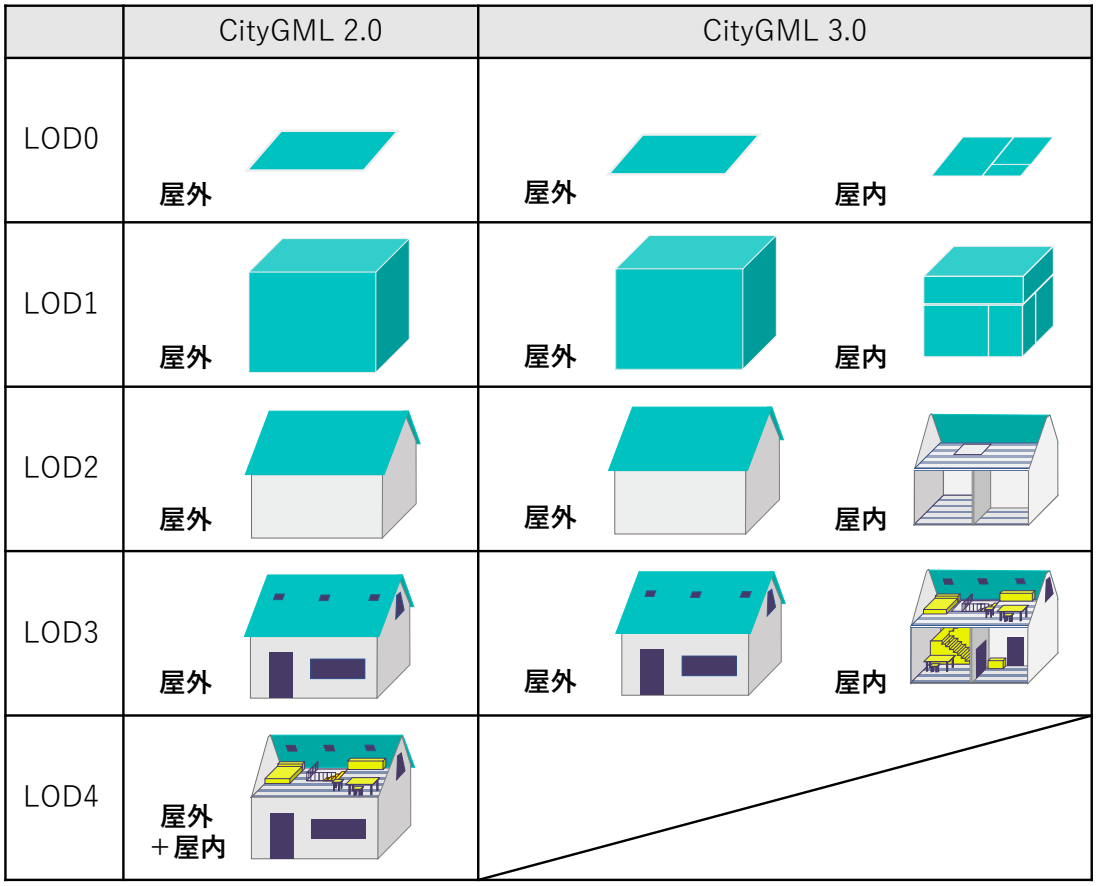



**図 2-3　CityGML 2.0 及び CityGML 3.0 における LOD の区分**

CityGML 3.0 における LOD の定義を表 2-2 に示す。

**表 2-2　CityGML 3.0 における LOD の定義**

| LOD | 定義 | Space の表現 | Boundary の表現 |
|---|---|---|---|
| LOD0 | 空間を、点、線又は面により表現する。境界を線又は面により表現する。面は、空間を平面に投影した結果である。線は、壁のような垂直の面を平面に投影した結果又は長い形状の地物の骨格となる。 | 点、線、面 | 線、面 |
| LOD1 | 空間は一律の高さを与えられた立体であり、境界は水平又は垂直の面となる。 | 立体 | 面 |
| LOD2 | 空間を線、面又は単一の立体として表現する。境界を面として表現する。LOD2 では詳細な形状は無視される。LOD2 における線は、アンテナや煙突のような長い空間の骨格として使用される。 | 線、面、立体 | 面 |
| LOD3 | 空間を、面又は単一の立体として表現する。境界を面として表現する。LOD3 は最も詳細な表現であり、記述可能な全ての幾何形状が対象となる。 | 線、面、立体 | 面 |

### 2.2.3.　空間属性の統合

CityGML 2.0 では、主題モジュールごとに各 LOD において各地物がもつべき空間属性を定義していた。例えば Building モジュールの場合、建築物の抽象概念である *_AbstractBuilding* には、LOD0 では面（*lod0RoofEdge*、*lod0FootPrint*）、LOD1 では立体（*lod1Solid*）というように、LOD0 から LOD4 までの空間属性を定義している。これに対し、CityGML 3.0 では、空間属性

は Core モジュールに定義された上位概念の地物にのみ空間属性を定義している。CityGML に定義された各都市オブジェクトは全て、上位概念の地物を継承するため、上位概念に定義された空間属性をもつことができる。

図 2-4 に空間属性が定義された概念モデルを示す。

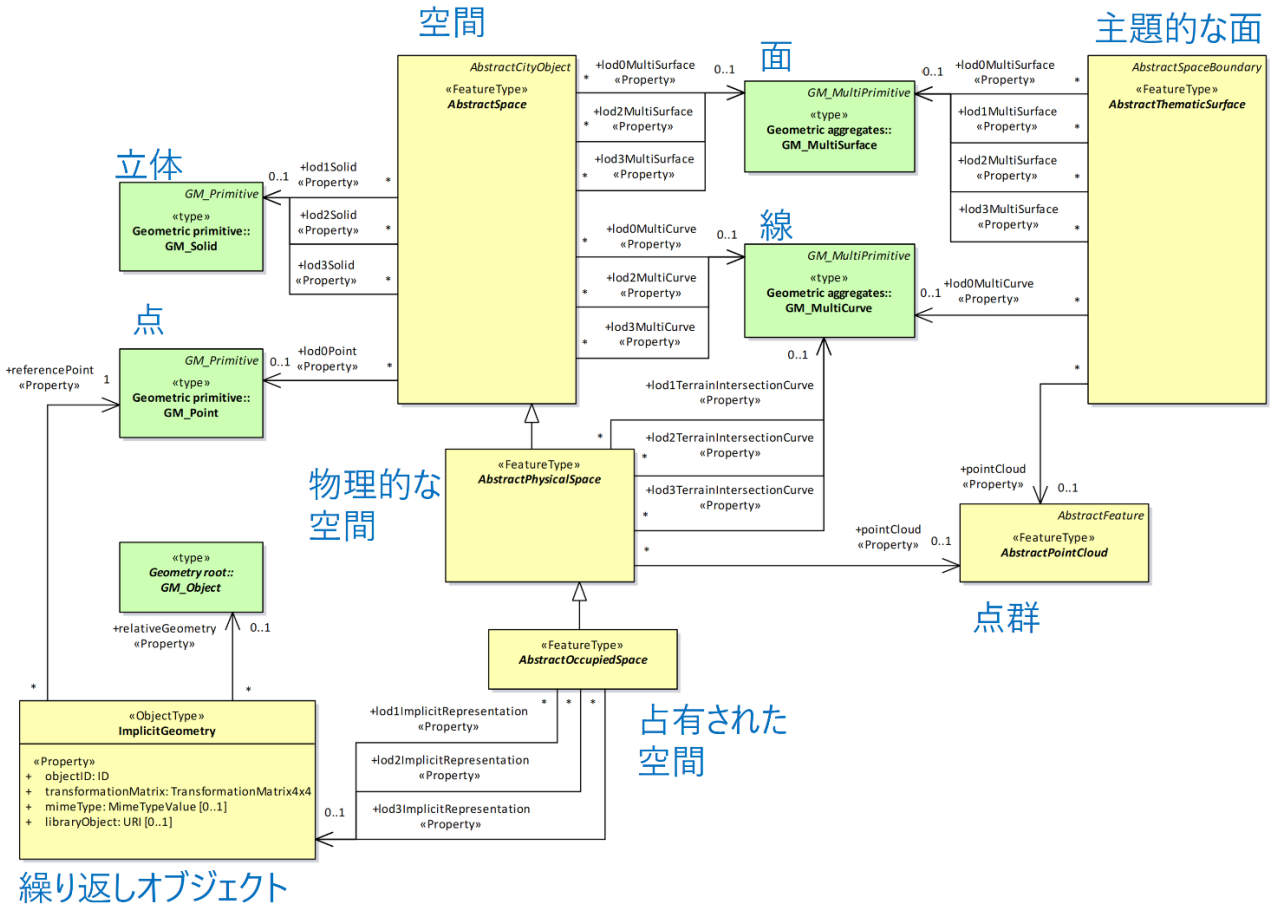



**図 2-4　CityGML 3.0 における空間属性の定義（出典[2] 一部補足）**

空間を示す *AbstractSpace* 及び空間の境界を示す *AbstractThematicSurface* には、LOD 別に空間属性が定義されている。例えば、*AbstractSpace* は、LOD0 では *lod0Point*、*lod0MultiCurve*、*lod0MultiSurface* というように、点、線又は面を空間属性としてもつことができる。また、LOD1 では *lod1Solid* として立体のみもつことができる。一方、*AbstractThematicSurface* は LOD0、LOD1、LOD2、LOD3 の各 LOD で面（*MultiSurface*）を空間属性としてもち、LOD0 の場合は線（*MultiCurve*）をもつこともできる。建築物のように、*AbstractSpace* を継承する地物は全て、LOD0 では *lod0Point*、*lod0MultiCurve*、*lod0MultiSurface* というように、点、線又は面のいずれかで記述できる。また、LOD1 では *lod1Solid* として立体でのみ記述可能となる。土地利用のように *AbstractThematicSurface* を継承する地物は全て、LOD0、LOD1、LOD2、LOD3 の各 LOD で面として記述でき、LOD0 の場合のみ線として記述することもできる。

空間属性を Core モジュールに定義することにより、これを継承する主題モジュールには以下の 2 点の影響がある。

- 各 LOD で記述可能な図形表現が増える。
- 各 LOD で使用可能な地物が増える。

例えば、建築物の場合、CityGML 2.0 では LOD0 では面のみ記述可能であったが、CityGML 3.0 では点、線、面のいずれでも記述できる。これは、建築物が *AbstractSpace* を継承しているからである。また、CityGML 2.0 では、LOD は地物の形状の詳細度だけではなくセマンティクスの詳細度を含んでいたが、CityGML 3.0 では LOD は形状の詳細度のみを表すと整理された。例えば、CityGML 2.0 では建築物の付属物（*BuildingInstallation*）は空間属性として *lod2Geometry*、*lod3Geometry* 及び *lod4Geometry* をもつ。また、開口部（*Window* 及び *Door*）は空間属性として *lod3MultiSurface* 及び *lod4MultiSurface* をもつ。これは、付属物は LOD2 以上、開口部は LOD3 以上で記述可能となることを意味する。しかし、CityGML 3.0 では付属物も *lod0Point*、*lod0MultiCurve* 又は *lod0MultiSurface* といった空間属性を継承し、開口部も *lod0MultiSurface* 又は *lod0MultiCurve* といった空間属性を継承するため LOD0 から付属物や開口部を記述できる。

幾何オブジェクト以外で都市オブジェクトの形状を表現することもできる。点群を使う方法と、繰り返しオブジェクトを使用する方法である。物理的な空間及び主題的な面は、幾何オブジェクトの代替として又は幾何オブジェクトと共に、点群（*AbstractPointCloud*）をもつことができる（関連役割 *pointCloud*）。また、占有された空間は、LOD1 から LOD3 の LOD では、実際の幾何オブジェクトの代替として繰り返しオブジェクト（*ImplicitGeometry*）により表現することができる（関連役割 *lod1ImplicitRepresentation*、*lod2ImplicitRepresentation* 及び *lod3ImplicitRepreesntation*）。*ImplicitGeometry* は、テンプレートとなるモデルと、そのテンプレートを配置したい位置や大きさ、向きを指定する変数との対であり、これを使用することで同じモデルを複数の都市オブジェクトの表現に繰り返し使用できる。

### 2.2.4. 地物の時間属性の追加と施工基準座標参照系の追加

CityGML 2.0 では、地物のデータが作成された日（*creationDate*）や削除された日（*terminationDate*）が定義されているが、地物そのものの発生や消失を表現する属性は定義されていなかった。これに対し、CityGML3.0 では、地物の時間的な性質（時間属性）が追加された。地物がいつ発生（*validFrom*）して、いつ消滅（*validTo*）したかを記述することが可能となり、このような属性をもつことができる地物として、*AbstractFeatureWithLifespan* が定義された（図 2-5 の青枠）。CityGML 3.0 に定義される全ての地物は、この *AbstractFeatureWithLifespan* を継承する。すなわち、全ての地物はこれらの時間属性をもつことができる。

また、都市モデル（*CityModel*）に適用される座標参照系として、施工基準座標参照系[1]（任意座標参照系）も記述できるようになった（図 2-5 の赤枠）。属性 *engineeringCRS* は、施工基準座標参照系を定義するための属性であり、施工基準座標参照系の原点、座標軸や単位等が記述できる。施工基準座標参照系の定義には、原点の地理座標等、地理座標参照系に変換するための情報を必ず含まなければならない。

なお、GML の仕様では、座標や幾何オブジェクト毎に座標参照系を指定できる。指定可能な座標参照系の種類は問わない。つまり、GML の仕様に従うと、一つの 3D 都市モデル内に、地理座標で記述された地物と、任意座標で記述された地物を混在させることができる。しかしながら、CityGML2.0 では、実装の観点から一つの都市モデル内では同一の空間参照系を使用することを強く推奨している。この推奨に従うと、地理座標で記述された地物と、任意座標で記述された地物は異なる都市モデル（データセット）に格納しなければならない。

CityGML 3.0 では、属性 *engineeringCRS* を使用して、都市モデルに対して明示的に施工基準座標参照系を指定できる。ただし個々の地物ではなく、都市モデル全体に対する属性であるため、CityGML 2.0 と同様、一つの都市モデルに異なる座標参照系で記述された地物を格納するという実装は想定されていないといえる。

---

[1] 施工基準座標参照系（engineering coordinate reference system）とは、施工基準原子（座標系の局所的な参照に対する関連付けを記述する原子）に基づく座標参照系（JIS X7111 座標による空間参照）であり、測地座標をもたない任意の原点からの相対位置により座標を記述する座標参照系のことである。プロジェクト対象地区など、局所的な範囲において使用される。

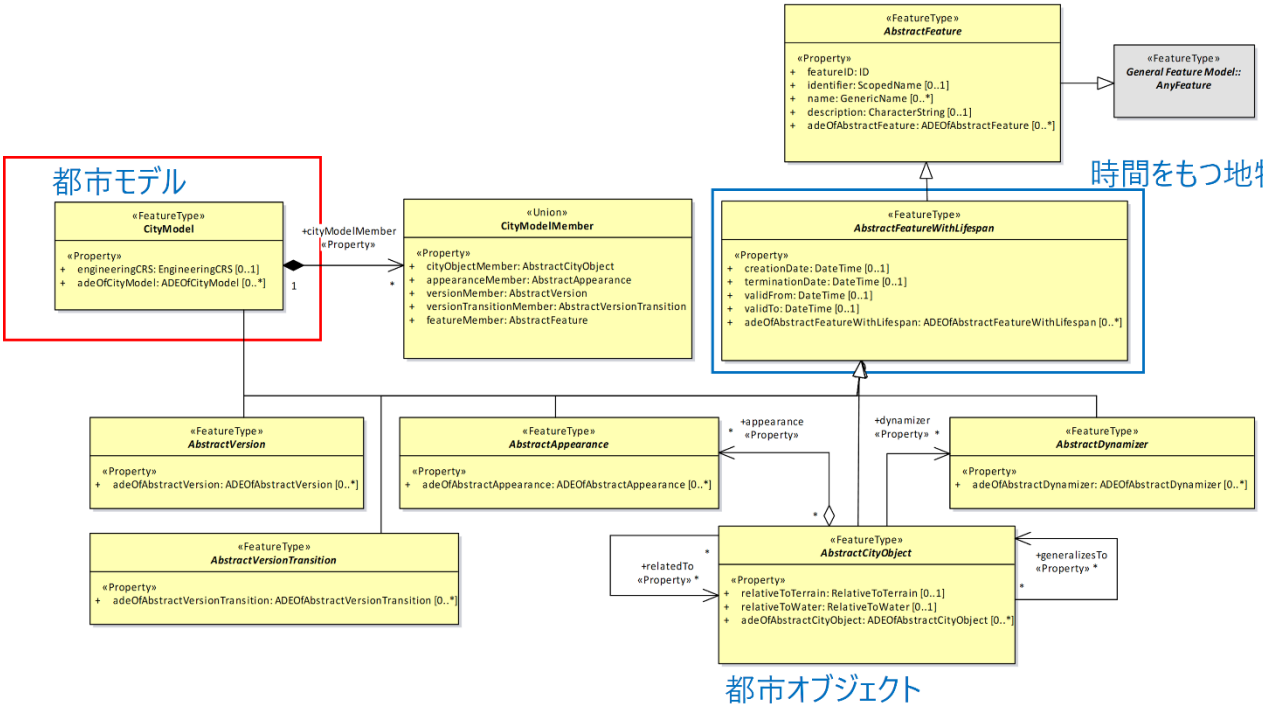



**図 2-5　City models and city objects（出典[2]一部補足）**

図 2-5 は、CityGML 3.0 における都市モデルとその構成要素との関係を示した概念モデルである。ISO 19109 応用スキーマのための規則が定める一般地物モデル（General Feature Model）の地物（*AnyFeature*）を継承し、CityGML における地物の概念（*AbstractFeature*）を定義する。さらに、時間的な概念が追加された地物（*AbstractFeatureWithLifespan*）を定義する。この *AbstractFeatureWithLifespan* を継承し、CityGML 3.0 に定義された各モジュールの上位概念が定義される。都市モデル（*CityModel*）には、その構成要素（関連役割 *cityModelMember*）として、都市オブジェクト（*AbstractCityObject*）、アピアランス（*AbstractAppearance*）、バージョン情報（*AbstractVersion*、*AbstractVersionTransition*：2.8 で説明）をもつ。また、都市オブジェクトは、関連役割 *appearance* 及び *dynamizer* により、アピアランス及びセンサーデータ（*AbstractDynamizer*）をもつことができる。さらに、都市オブジェクトは、関連役割 *relatedTo* によって関連する他の地物を明示することや、関連役割 *generalizesTo* により、異なる LOD で記述された同じ地物と関連づけることができる。

## 2.3.　Generics モジュール

Generics モジュールには、主題モジュールで定義済みの地物以外の地物を追加したい場合や、主題モジュールに定義された地物に属性を追加したい場合に使用する、汎用的な地物と属性が定義される。

CityGML 3.0 では、地物として、論理的な空間、占有空間、非占有空間、及びこれらの境界となる主題面が定義されたため、各々に対応する汎用地物（*GenericLogicalSpace*、*GenericOccupiedSpace*、*GenericUnoccupiedSpace* 及び *GenericThematicSurface*）が追加された（図 2-6）。Generics モジュールを使用して地物を追加する場合は、その地物の特性を踏まえ、適切な汎用地物を選択する必要がある。

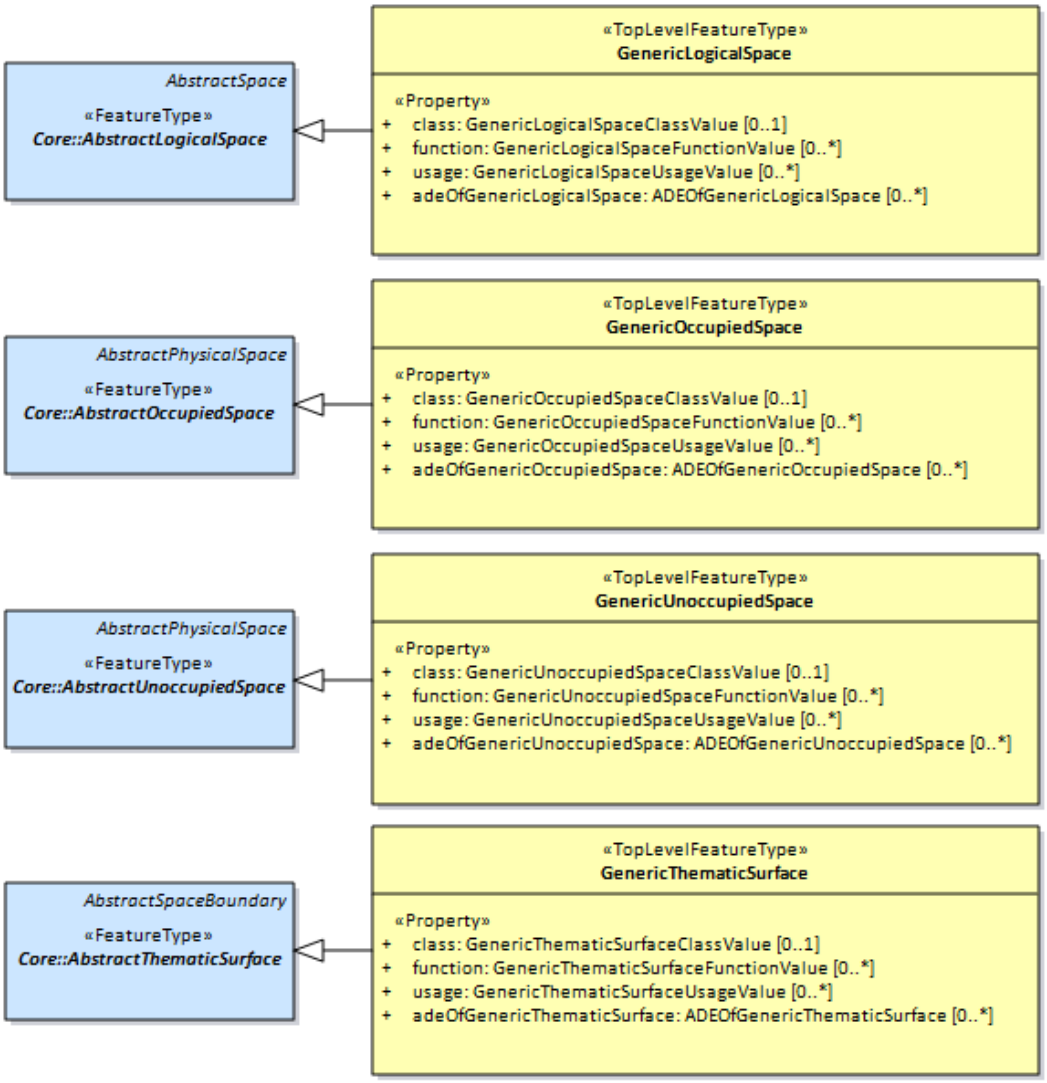



**図 2-6　汎用都市オブジェクト（出典[2] ©OGC）**

また、汎用属性として、新たに属性の型がコード型となる汎用属性（*CodeAttribute*）が追加された（図 2-7）。

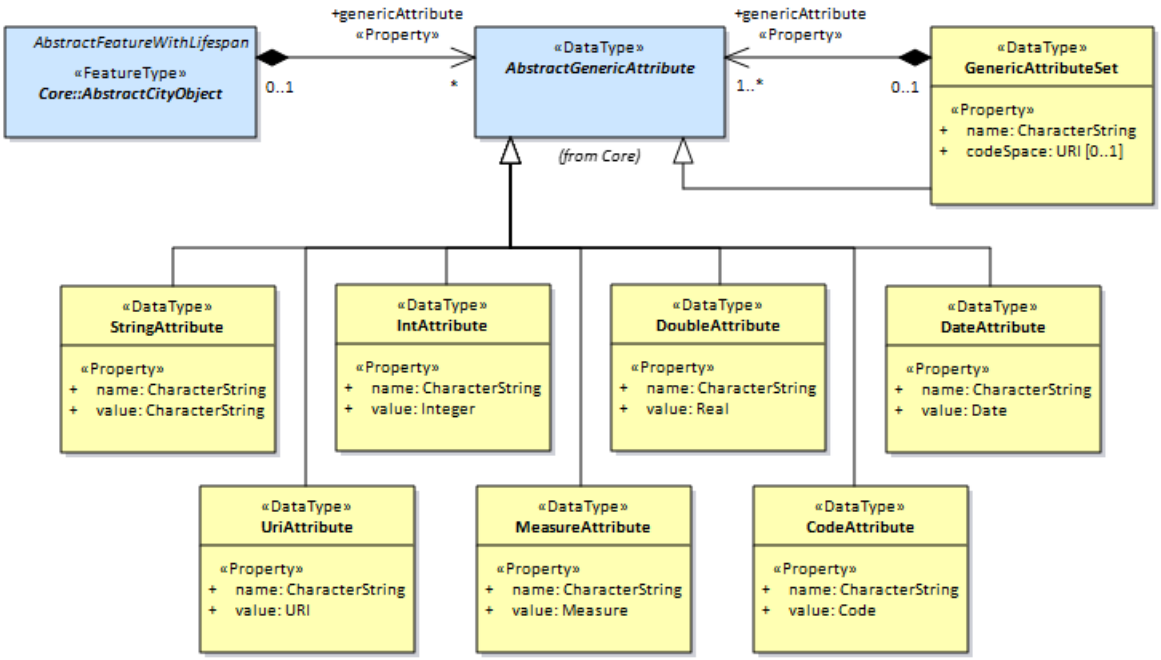



**図 2-7　汎用属性（出典[2] ©OGC）**

## 2.4.　Building モジュール

Building モジュールには、建築物（*Building*）及びこれの部品となる地物が定義される。

CityGML 3.0 では、BIM（Building Information Modeling）の標準である IFC（Industry Foundation Classes）や欧州における空間情報基盤（INSPIRE）の 3 次元建築物モデルとの整合の観点から改定が行われている。また、*Building*、*Bridge* 及び *Tunnel* の上位概念として、新たに Construction モジュールが追加され、*Building*、*Bridge* 及び *Tunnel* に共通する概念である境界面（屋根面や壁面等）や開口部（窓及び扉）は、Construction モジュールに定義されることとなり、Building モジュールからは境界面及び開口部は削除された。Construction モジュールについては、2.6 にて説明する。

Building モジュールの概念モデルを図 2-8 に示す。建築物の抽象概念である *AbstractBuilding* を継承し、建築物（*Building*）及び建築物部分（*BuildingPart*）が定義される。建築物は、建築物部分をもつことができる（関連役割 *buildingPart*）。建築物及び建築物部分は、構成要素として Building モジュールに定義された他の地物をもつことができる。具体的には、部屋（*BuildingRoom*）、区画（*BuildingUnit*）及び階（*Storey*）、付属物（*BuildingInstallation*）、家具（*BuildingFurniture*）及び建築部材（*BuildingConstructiveElement*）である。部屋はその構成要素として、付属物や家具をもつことができる。区画及び階も、構成要素として部屋、付属物及び家具をもつことができる。加えて、区画はその構成要素として階をもってもよいし、階はその構成要素として区画をもってもよい。このように、Building モジュールは、建築物である *Building* が全体を示す地物であり、その構成要素として様々な地物が存在し、階層的なデータ構造として定義されている。

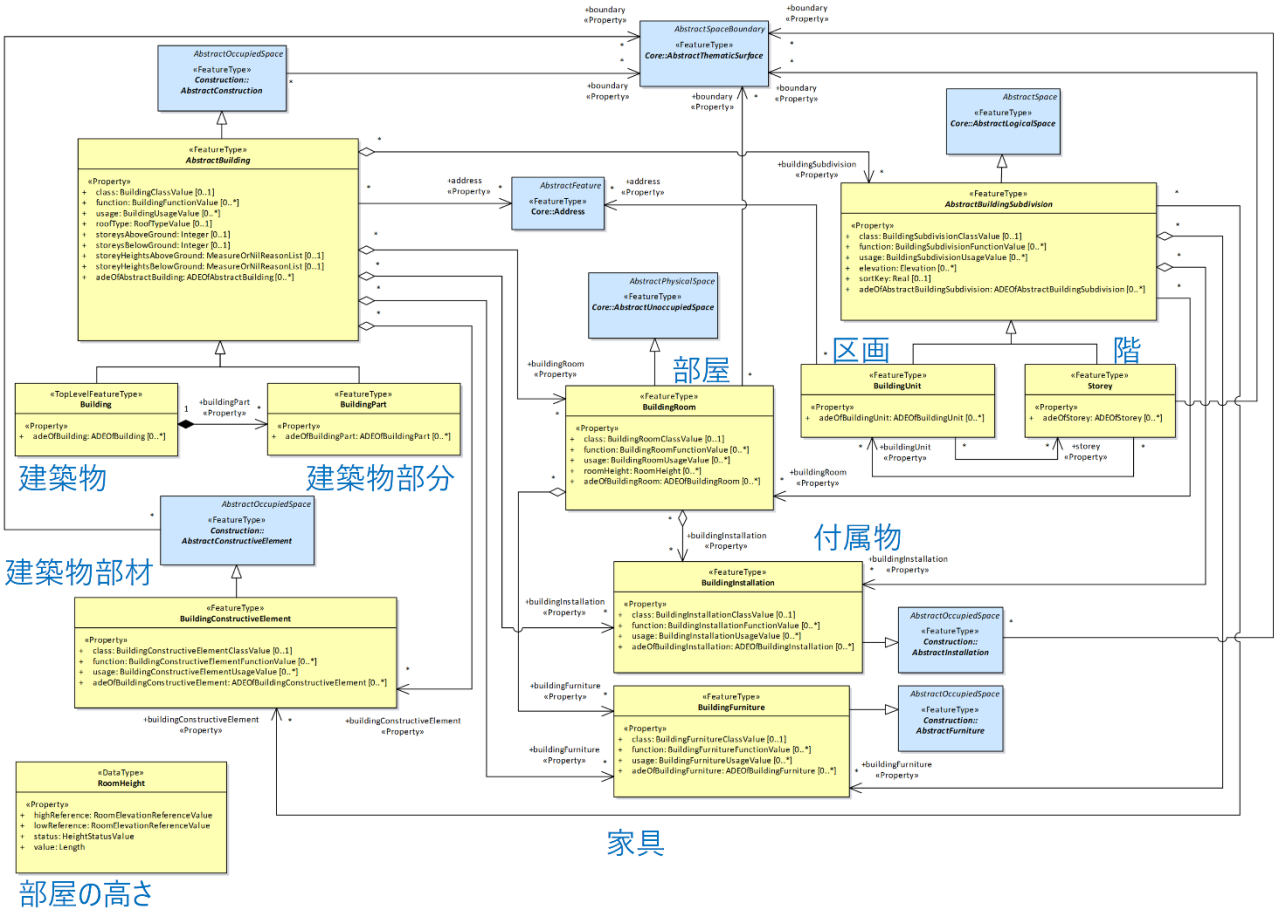



**図 2-8　Building モジュールの概念モデル（出典[2]一部補足）**

CityGML 3.0 における Building モジュールの改定点を以下に示す。

(1)　BuildingUnit 及び Storey の追加

屋内空間をより詳細に区分する概念（*AbstractBuildingSubdivision*）が追加され、これを継承する具象クラスとして *BuildingUnit* 及び *Storey* が定義された[2]。これらは、論理的な空間である *AbstractLogicalSpace* を継承する。*BuildingUnit* は、例えば防火区画のように、建築物の一部を区切るための地物である。*Storey* は、建築物の階に該当する。

*BuildingUnit* は複数の *Storey* から構成してもよく（関連役割 *storey*）、階を複数の区画から構成してもよい（関連役割 *buildingUnit*）。*BuildingUnit* に含まれる *Storey* の中に *BuildingUnit* を作成するというような記述も可能である（図 2-9）。

[2] CityGML 2.0 では建築物内部の階や区画の表現に特化した地物は定義されておらず、グループ化の仕組みである *CityObjectGroup* を用いて表現することとなっていた。

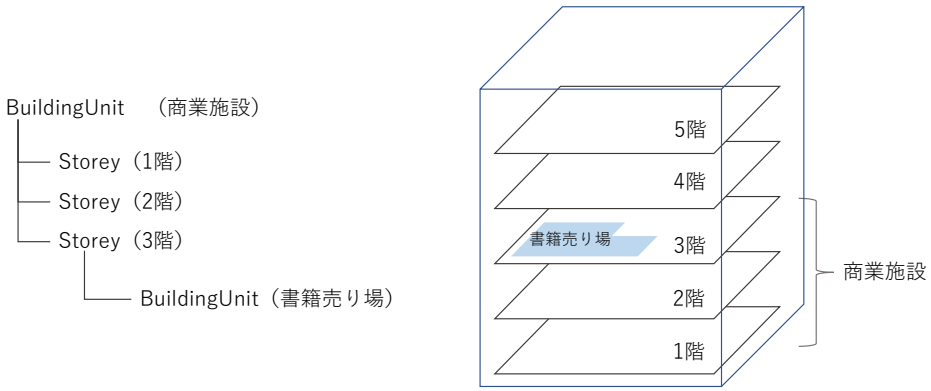



**図 2-9　*BuildingUnit* 及び *Storey* の例**

(2)　BuildingConstructiveElement の追加

建築物を構成する建築部材を表現する概念として、*BuildingConstructiveElement* が追加された。この地物を適用する例として、壁、スラブ、階段及び梁が挙げられている。

ここでの壁は、*WallSurface*（壁面）を境界とする、厚みをもった壁を指す（図 2-10）。CityGML 2.0 では、建築物や部屋の境界面となる壁面しか定義されておらず、厚みをもった壁を地物として表現できなかった。CityGML 3.0 では、*BuildingConstructiveElement* を使用し、壁を地物として表現できる。

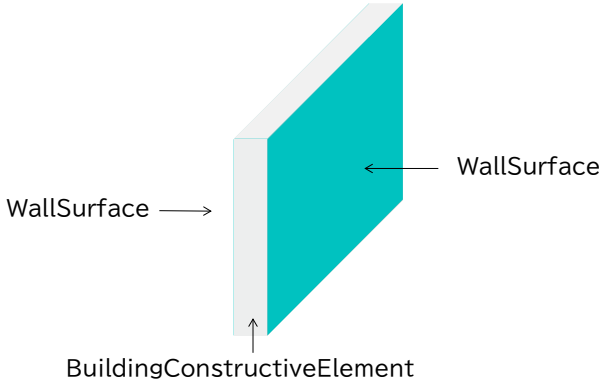



**図 2-10　*WallSurface*（壁面）と *BuildingConstructiveElement* として表現する壁**

*BuildingConstructiveElement* は、IFC で定義された *IfcWall*、*IfcSlab* 及び *IfcBeam* などの建築物を構成する部材を示すクラスと対応づけるために追加された地物である。IFC では壁は立体として表現されるが、前述したように CityGML 2.0 では壁の表面のみが個別の地物として表現される。そのため、IFC から変換する場合には立体として表現された壁から、境界面に対応する面を抽出する必要があった。CityGML 3.0 では *BuildingConstructiveElement* を使用することで、IFC から CityGML にマッピングすることが容易となる（図 2-11）。

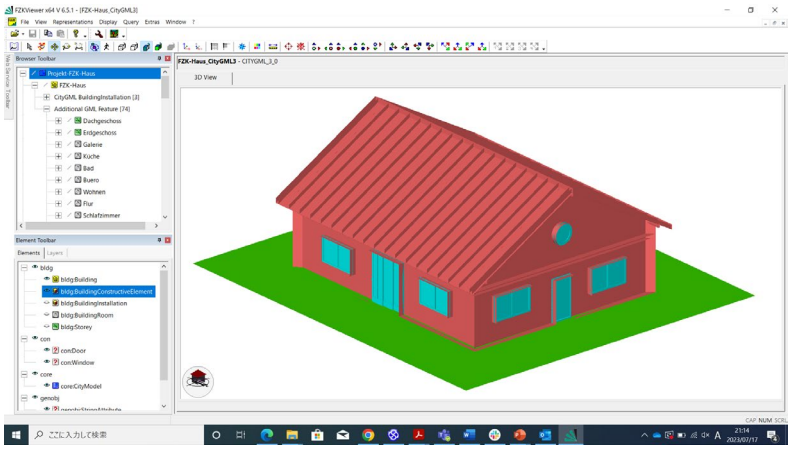


**図 2-11　IFC クラスからの変換により作成された *BuildingConstructiveElement***

ただし、建築物や部屋の境界面を地物として表現したい場合には、CityGML 2.0 と同様、立体である壁や床等のオブジェクトから境界面を抽出する必要がある。例えば、3D 都市モデル整備のための BIM 活用マニュアル[10]では、BIM モデルにおける部屋に相当する空間である IfcSpace を使用し、IfcSpace を構成する境界面を壁面や床面として抽出する方法を提案している。また、簡素化する方法として、壁や床等の中心となる面を作成し、部屋の連続性を表現する方法[11]もある（図 2-12）。

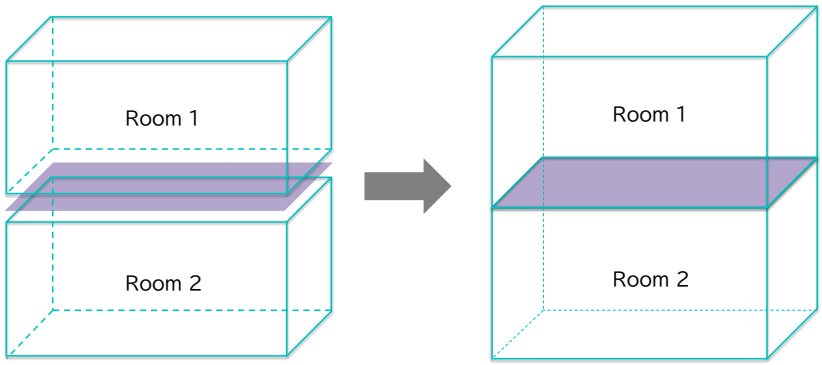



**図 2-12　中心面により部屋の連続性を表現する例**

なお、付属物（*BuildingInstallation*）と *BuildingConstructiveElement* が対象とする地物は CityGML 3.0 では明確に定義されていない。例えば、階段を表現する場合、CityGML 2.0 では付属物（*BuildingInstallation*）を使用するが、CityGML 3.0 では *BuildingInstallation* 又は *BuildingConstructiveElement* を使用しても表現することができる。どのような場合に、あるいはどのような階段の場合にはどちらを使用するのかといった統一的なルールが無ければ、作成者によって使用する地物に差異が生じる。

(3)　BuildingInstallation 及び IntBuildingInstallation の統合

CityGML 2.0 では、屋外の付属物は *BuildingInstallation*、屋内の付属物は *IntBuildingInstallation* と地物として分けて定義されていたが、CityGML 3.0 では、*BuildingInstallation* に統合された。*BuildingInstallation* の上位概念となる *AbstractInstallation* には属性 *relationToConstruction* があり、この属性でその付属物が外部に存在するのか、内部に存在するのか又は両方なのかを明示的に区別することができる。

(4)　RoomHeight の追加

CityGML 3.0 では、部屋の属性として新たに部屋の高さ（*roomHeight*）が追加され、この部屋の高さを表すデータ型として

*RoomHeight* が定義された。*RoomHeight* は、2 点（*highReference*、*lowReference*）を指定し、2 点間の距離を高さ（*value*）として格納できるデータ型である。さらに、この値は計測値であるのか、あるいは推定値であるのかという状態（*status*）を記録できる。*highReference* 及び *lowReference* はコードリストにより指定する必要があるが、コードリストは CityGML 3.0 では定義されていない。

## 2.5. Transportation モジュール

Transportation モジュールは、道路（*Road*）や鉄道（*Railway*）などの交通に関する地物を定義するモジュールである。

CityGML 3.0 に新たに空間と境界とが定義されたことにより、概念モデルが大きく改定された。また、CityGML 2.0 の Transportation モジュールに対する課題として以下が挙げられており、これを解決する改定が行われている[14]：

- LOD0 のネットワークが、道路単位なのか車線単位なのか明確ではない。
- LOD0 のノードに該当する地物が定義されていない。
- 道路の LOD0 は面として表現されるが、何が含まれているか明確ではない。
- 道路などの交通オブジェクトを区切る地物がない。
- 交差点やロータリーを明示的に記述できない。

Transportation モジュールの概念モデルを図 2-13 に示す。

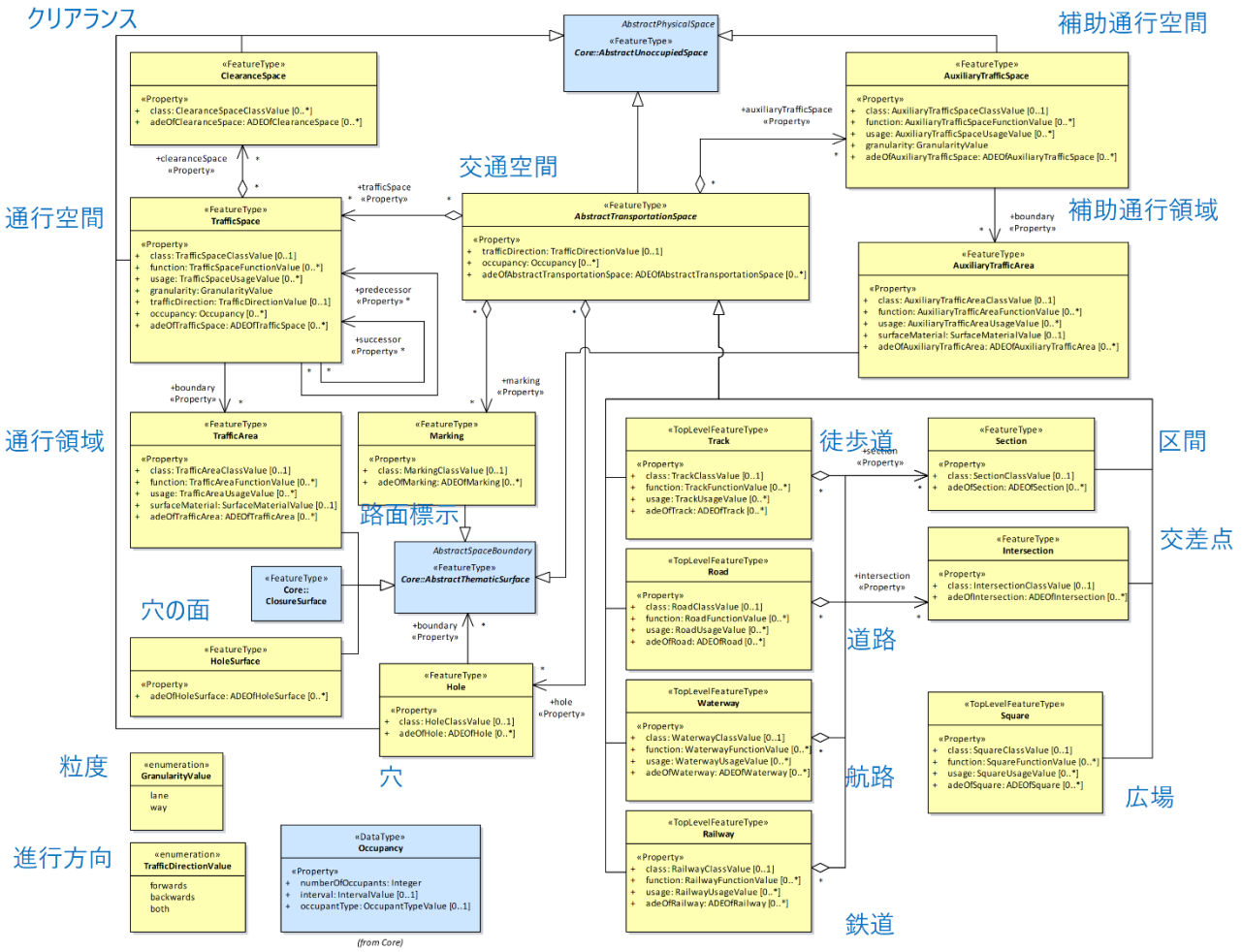



**図 2-13　Transportation モジュールの概念モデル（出典[2]一部補足）**

*AbstractTransportationSpace* は、交通のための空間を示す地物であり、これを継承して徒歩道（*Track*）、道路（*Road*）、航路（*Waterway*）、鉄道（*Railway*）及び広場（*Square*）が定義される。すなわち、*Track*、*Road*、*Waterway*、*Railway* 及び *Square* は「空間」であり、*AbstractTransportationSpace* は *AbstractUnoccupiedSpace* を継承しているため、「非占有空間」となる。これらの地物が Transportation モジュールにおける<<TopLevelFeatureType>>であり、全体を示す地物となる。これら以外の地物は、その構成要素として定義されている。*AbstractTransportationSpace* の構成要素として、通行可能な空間を示す *TrafficSpace* 及びこの機能を補助する *AuxiliaryTrafficSpace* がある。これらは空間であり、空間を区切る面として、*TrafficArea* 及び *AuxiliaryTrafficArea* が定義されている。また、通行空間には、障害物がなにもない空間であるクリアランス（*ClearanceSpace*）を設けることができる。さらに、交通空間はその構成要素として路面標示（*Marking*）や穴（*Hole*）を含めてもよい。また、交通空間を継承する地物として区間（*Section*）及び交差点（*Intersection*）がある。これらは常に徒歩道、道路、航路又は鉄道のいずれかの構成要素としてのみ存在する（関連役割 *section* 及び *intersection*）。

Transportation モジュールも Building と同様に、徒歩道、道路、航路、鉄道及び広場が全体となり、その構成要素として通行空間や補助通行空間があり、その境界面として通行領域や補助通行領域が存在する、という階層的なデータ構造となる。

Transportation モジュールの改定内容を以下に示す。

(1)　空間を示す地物の追加

道路や鉄道の空間を示す地物として *TrafficSpace*(通行空間)、*AuxiliaryTrafficSpace*(補助通行空間)及び *ClearanceSpace*（クリアランス空間）が追加された（図 2-14）。CityGML 2.0 では、道路面や鉄道の線路を表す *TrafficArea*（通行領域）及び *AuxiliaryTrafficArea*（補助通行領域）が定義されていたが、これらは通行空間及び補助通行空間を区切る境界面となる。また、*ClearanceSpace* は、通行空間上空の障害物のない空間を指し、ドローンなどの移動体が通行可能な空間として利用されることを想定して CityGML 3.0 で新たに定義されている。

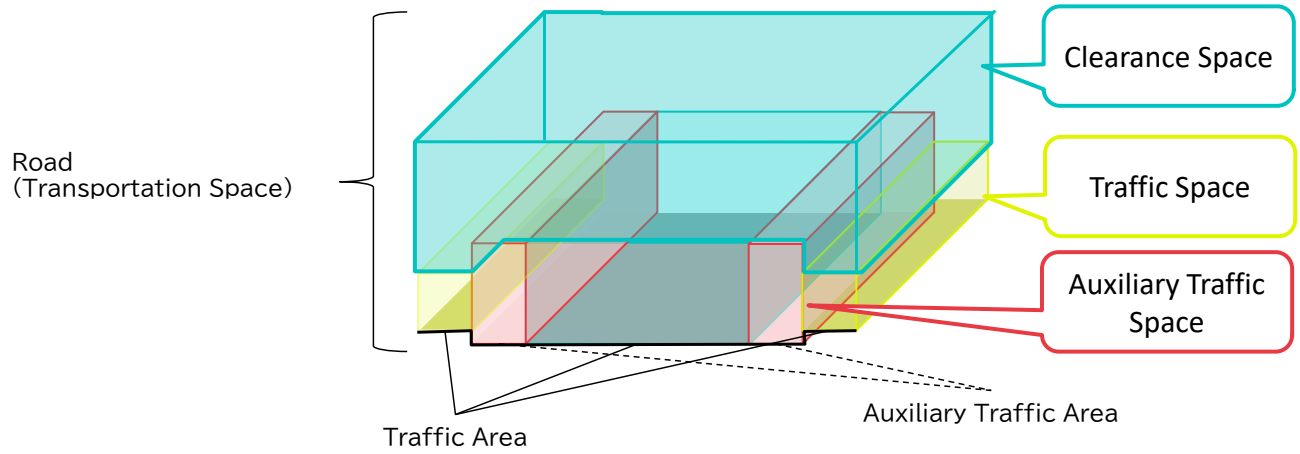



**図 2-14　*TrafficSpace*、*TrafficArea* 及び *ClearanceSpace* の例**

通行可能な空間を示す *TrafficSpace* には、分類（*class*）、機能（*function*）及び用途（*usage*）といった交通空間を細分する属性に加え、通行に関する属性や関連が追加された。属性 *granularity* は、この交通空間が道路単位なのか、あるいは車線単位なのかを示す情報の粒度を示すフラグである（図 2-15）。CityGML 2.0 では、LOD0 及び LOD1 において道路のような交通オブジェクトを表現し、LOD2 以降で車道、歩道、路肩といった詳細な区分を *TrafficArea* や *AuxiliaryTrafficArea* を使用して表現可能であった。CityGML 3.0 では、LOD はあくまで空間属性の詳細度の区分であり、情報の粒度は LOD の概念には含まれない。

属性 *trafficDirection* は、この通行空間を通行可能な向き（前、後、双方向）を示す。

また、通行空間は、通行可能な他の交通空間との関連をもつ。関連役割 *predecessor* はこの通行空間の前にある通行空間、*successor* は、この通行空間の後にある通行空間である（図 2-16）。これらの関連役割は、通行空間の連続性を意味的に表現するために使用する。さらに、属性 *occupancy* は、この交通空間が車両等により占用されている場合にその内容を記すことができる。

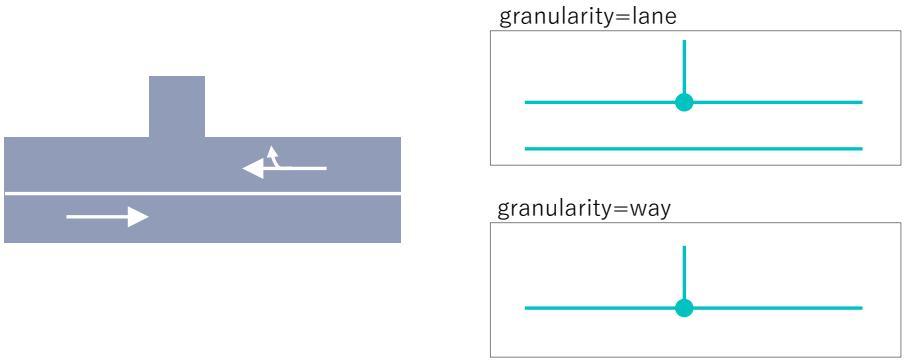
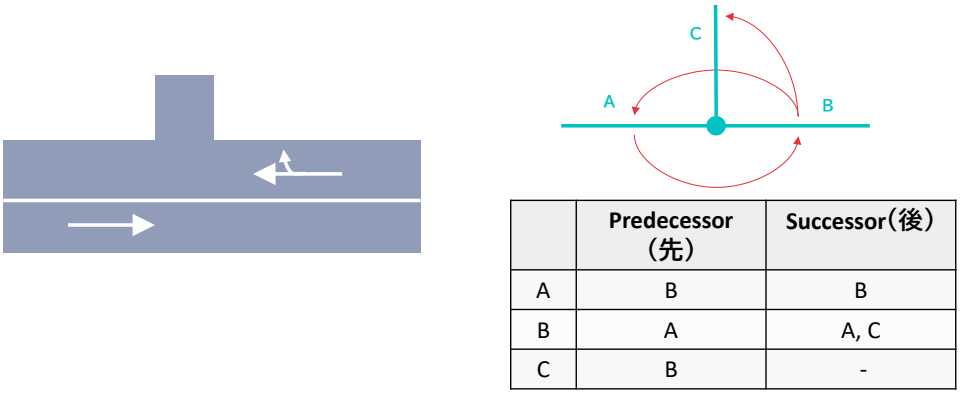



**図 2-15　情報の粒度**




| | Predecessor（先） | Successor（後） |
|---|---|---|
| A | B | B |
| B | A | A, C |
| C | B | - |

**図 2-16　意味的なネットワーク表現（出典[12]）**

(2)　区間および交差点を示す地物の追加

CityGML 3.0 では、道路や鉄道を区間（*Section*）に区切ることができるようになった。また、複数の道路や鉄道が交わる交差点（*Intersection*）も表現可能となる。道路や鉄道は、区間及び交差点の集まりとして表現できる。この区間や交差点は、*TrafficSpace* 及び *AuxiliaryTrafficSpace* の集まりとして表現され、さらに、*TrafficSpace* 及び *AuxiliaryTrafficSpace* の境界面が *TrafficArea* 及び *AuxiliaryTrafficArea* として区分されるという階層構造になる。

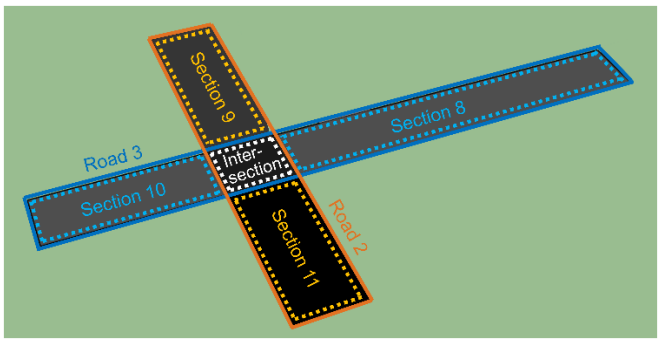



**図 2-17 *Section* 及び *Intersection* の例（出典[4]　©OGC）**

図 2-17 は道路と道路を構成する区間及び交差点の例である。Road 2 は、Section 9、Intersection 及び Section 11 の区間又は交差点から構成される。このとき、Road 2 と交差する Road 3 は、Section 10、Intersection 及び Section 8 の区間又は交差点から構成される。Road 2 及び Road 3 道路が交差する交差点である Intersection は Road 2 と Road 3 によって共有される。

(3)　路面標示及び穴を示す地物の追加

CityGML 3.0 では、交通モジュールに新たな地物として、路面標示を示す *Marking*、道路上の穴として *Hole* 及び *HoleSurface* が定義された。*Hole* は穴の空間であり、*HoleSurface* は道路面に存在する穴の表面である。*Hole* 及び *HoleSurface* はマンホール、道路の損傷又は側溝の記述に適用できる。また、*Marking* は、交通の構造や制限のために示される路面標示を表現したい場合に使用できる。

これらのマンホールや路面標示は、交通モジュールの地物として記述することは必須ではなく、都市設備（*CityFurniture*）として記述してもよい。また、交通モジュールの *HoleSurface* や *Marking* として記述しつつ都市設備（*CityFurniture*）として記述し、両者には関係がある（*relatedTo*）とすることもできる（2.2.4 参照）。

**図 2-18　Transportation モジュールのオンラインデモ**

**(https://wiki.tum.de/display/gisproject/Online+Demo+Collection)**

CityGML 3.0 に従った道路のデータは、ミュンヘン工科大学が作成したオンラインデモ（https://wiki.tum.de/display/gisproject/Online+Demo+Collection）より確認できる（図 2-18）。ミュンヘン市では、車線レベルの道路データを収集しており、収集したデータを CityGML 形式に変換したうえで 3DCityDB-Web-Map-Client を使用して可視化している。

(4)　航路の追加

CityGML 3.0 では、新たに船舶が交通する空間を表現するための地物として *Waterway* が追加された。道路や鉄道と同様の階層構造をもつことができる。

## 2.6.　Construction モジュール

Construction モジュールは、CityGML 3.0 で新たに追加された主題モジュールであり、建築物、橋梁及びトンネルの上位概念として定義された。建築物、橋梁及びトンネルの各モジュールに定義された地物は、Construction モジュールに定義された地物を継承し、定義される。また、Construction モジュールには、建築物、橋梁及びトンネルに共通的な地物として、境界面や開口部が定義されている。さらに、*OtherConstruction* として、橋梁やトンネル以外の構造物の記述に使用できる地物が新たに定義された。*OtherConstruction* は、例えばダムや堤防、防波堤などの構造物の記述に使用できる。

図 2-19 に Construction モジュールの概念モデルを示す。

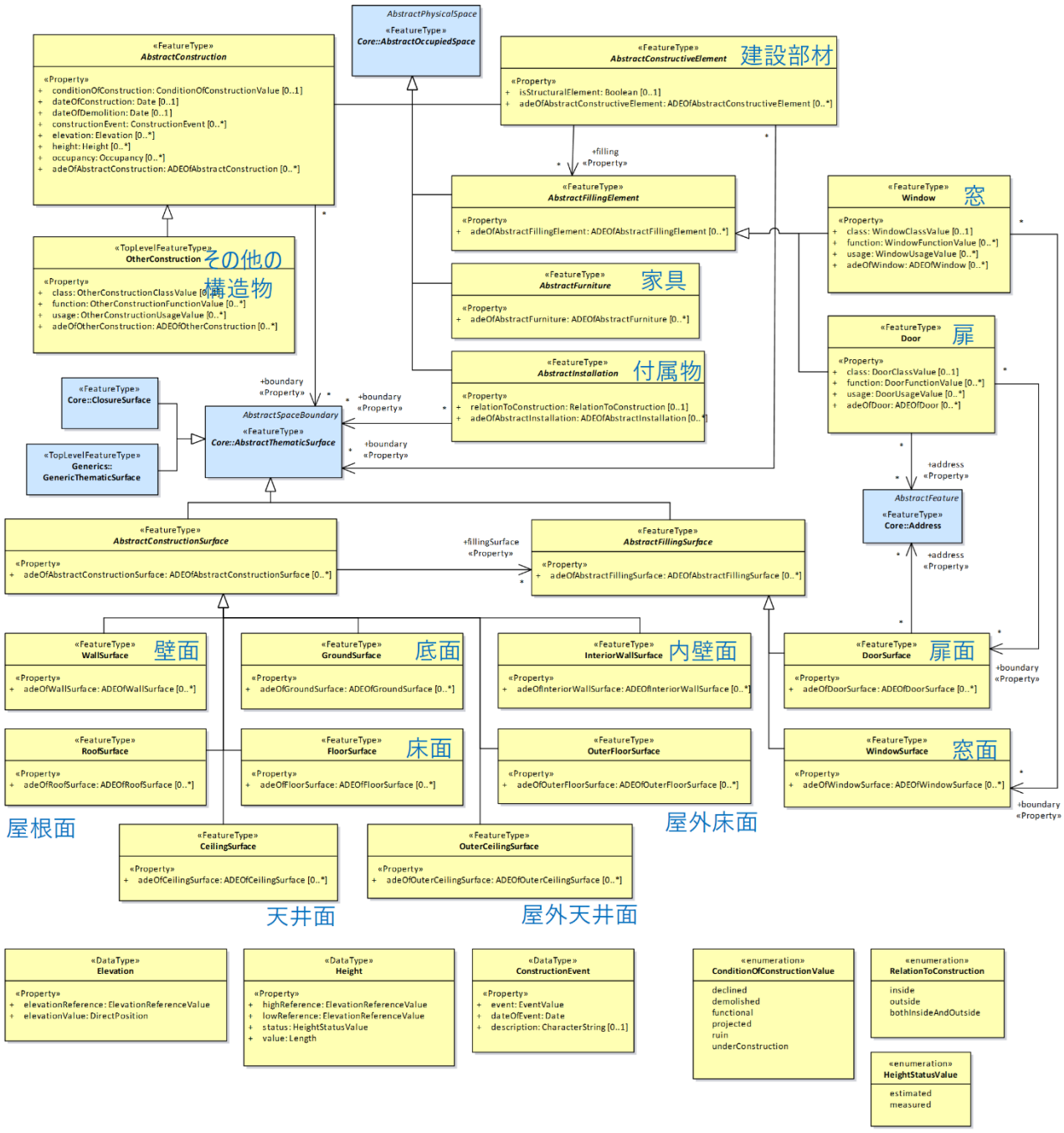



**図 2-19　Construction モジュールの UML クラス図（出典[2]一部補足）**

Construction モジュールには、構造物の抽象概念として、*AbstractConstruction* が定義されている。また、構造物の部材である *AbstractConstructiveElement*、固定的な設備（付属物）である *AbstractInstallation*、可動の設備である *AbstractFurniture* が定義されている。これらの地物は全て抽象クラスであるため、建築物、橋梁及びトンネルモジュールでは各々を継承し、*BuildingConstructiveElement*、*BuildingInstallation* 及び *BuildingFurniture* のような具象クラスを定義している。また、CityGML 3.0 では新たに *AbstractFillingElement* が定義された。これは、建設部材である *AbstractConstructiveElement* に埋め込まれる地物であり、具体的には窓（*Window*）及び扉（*Door*）となる。

構造物、付属物及び建設部材の境界面は、*AbstractThematicSurface* を継承して、*AbstractConstructionSurface* 及び *AbstractFillingSurface* として定義されている。これらは抽象クラスであるため、*AbstractConstructionSurface* には、*WallSurface*（壁面）や *RoofSurface*（屋根面）などが、また、*AbstractFillingSurface* には *WindowSurface*（窓面）及び *DoorSurface*（扉面）が定義されている。境界面や開口部の考え方は CityGML 2.0 と同様であるが、CityGML 2.0 ではこれらの地物が建築物、橋梁及びトンネルの各モジュールで定義されていたのに対し、CityGML 3.0 では、Construction モジュールにのみ定義され、建築物、橋梁及びトンネルの各モジュールから共通的に利用されるようになった点が異なる。

構造物である *AbstractConstruction* には、表 2-3 に示す属性が定義されている。

**表 2-3　*AbstractConstruction* の属性**

| 属性名と多重度 | 概要 |
|---|---|
| *conditionOfConstruction* [0..1] | 構造物のライフサイクルにおける状況を示す。 |
| *dateOfConstruction* [0..1] | 完成日。 |
| *dateOfDemolition* [0..1] | 取り壊し日。 |
| *constructionEvent* [0..*] | 構造物の建設に係るイベントの記録。 |
| *elevation* [0..*] | 構造物の標高。 |
| *height* [0..*] | 構造物の高さ。 |
| *occupancy* [0..*] | その構造物を占有する人や物の情報。 |
| *adeOfAbstractConstruction* [0..*] | 構造物の拡張用の属性。 |

*conditionOfConstruction* は当該建築物が現在どのような段階（計画、施工中、完成等）にあるのかを記述するための属性であり、各工程の記録は *constructionEvent* として記述できる。また、一つの構造物は、構造物の標高（*elevation*）や高さ（*height*）を複数もつことができる。標高は計測する地点の種類とその地点での標高との対であり、高さは高い地点と低い地点の 2 点の種類とその距離により表現される。標高又は高さを測る地点の種類を示すコードリストは CityGML 3.0 では定義されていないが、この考え方は INSPIRE の標準仕様を参照していることから、参考として図 2-20 に INSPIRE の標準仕様に示されたコードリストを示す。

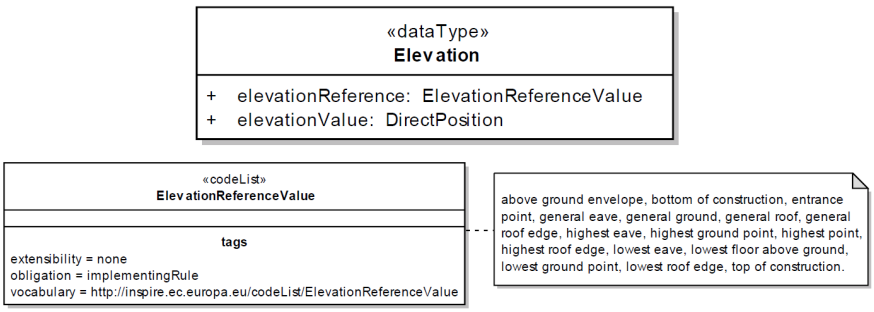



**図 2-20　*ElevationReferenceValue*　（出典[15]）**

さらに属性 *occupancy* には、構造物を占有する人や物の情報を格納できる。

建築物、橋梁、トンネル及びその他の構造物は、*AbstractConstruction* を継承することから、上述した属性を全て持つこと

ができる。

## 2.7. Dynamizer モジュール

Dynamizer モジュールは、CityGML 3.0 において新たに追加されたモジュールであり、地物の属性（空間、主題及びアピアランスを含む）が時間経過に伴い変化するという表現を可能とし、都市モデルとセンサーデータを統合することを目的として開発されたモジュールである（図 2-21）。

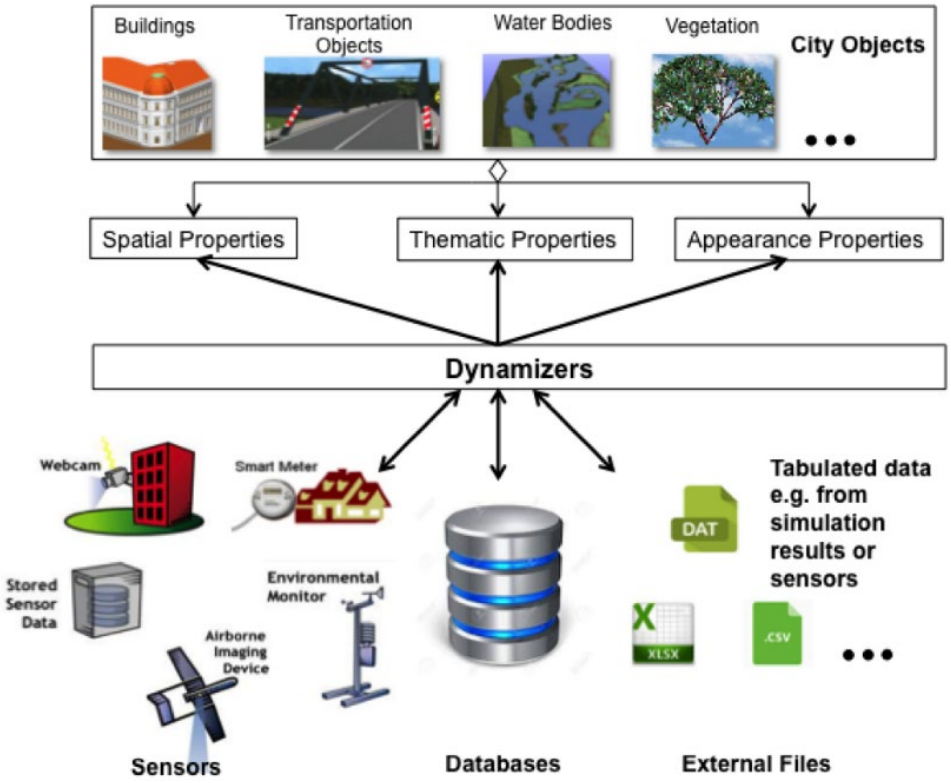



**図 2-21 Dynamizer のコンセプト（出典[2] ©OGC）**

地物の時間的な変化の表現を行うための地物は、CityGML 2.0 では定義されていなかったが、CityGML 3.0 では時間的な変化を表現するために、“Versioning”と“Dynamizer”の 2 つのモジュールが新たに追加されている。“Versioning”が、建物の新築や取り壊し、都市モデルの複数バージョンの管理など、その変化がゆっくりとした質的な変化を管理するためのモジュールであるのに対し、“Dynamizer”は、以下に示すような、高頻度な又は動的な変動を表す属性の量的な変化を管理することを目的として設計されたモジュールである。

- 地物の形状や位置の変化のような、空間属性の変化
- エネルギー需要、温度、日射量、交通密度、汚染濃度、建物の壁への過圧などの主題属性の変化
- シミュレーション又は測定により得られるセンサー又はリアルタイムデータの変化

Dynamizer モジュールの概念モデルを図 2-22 に示す。

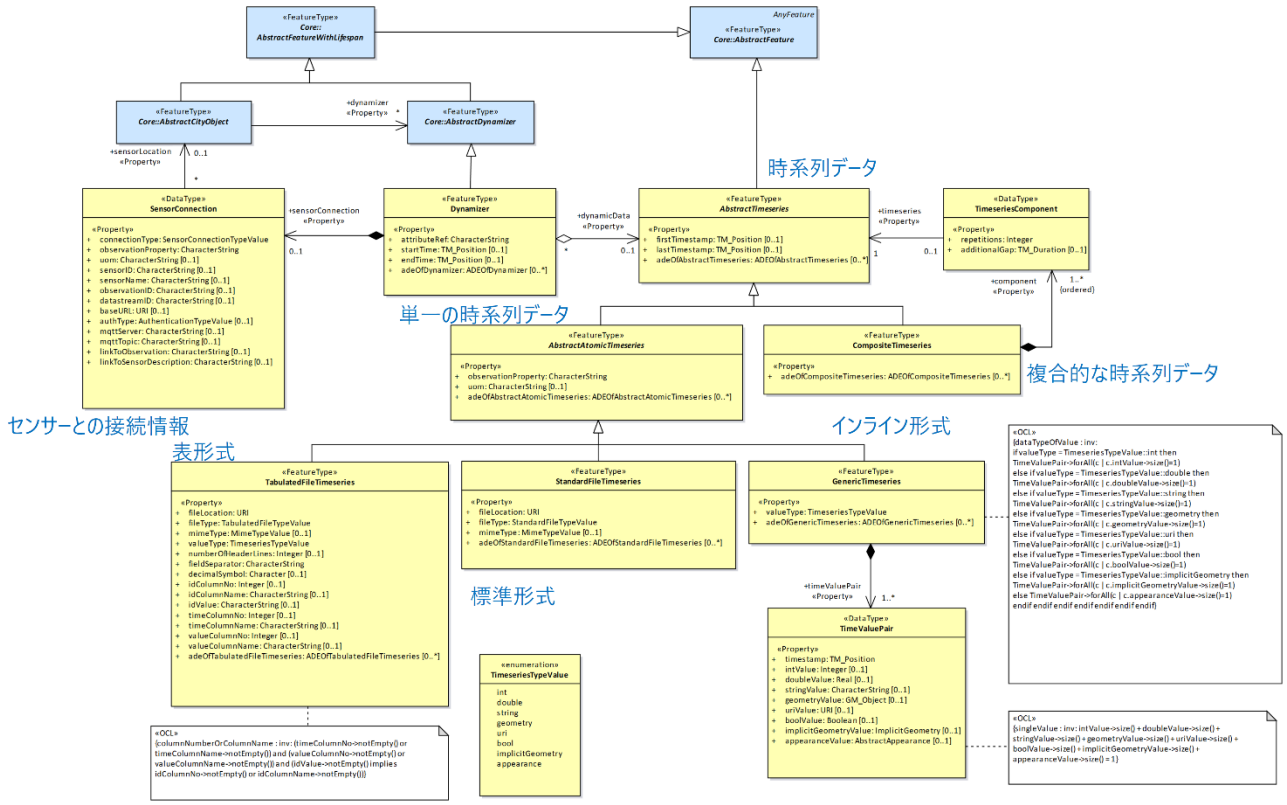
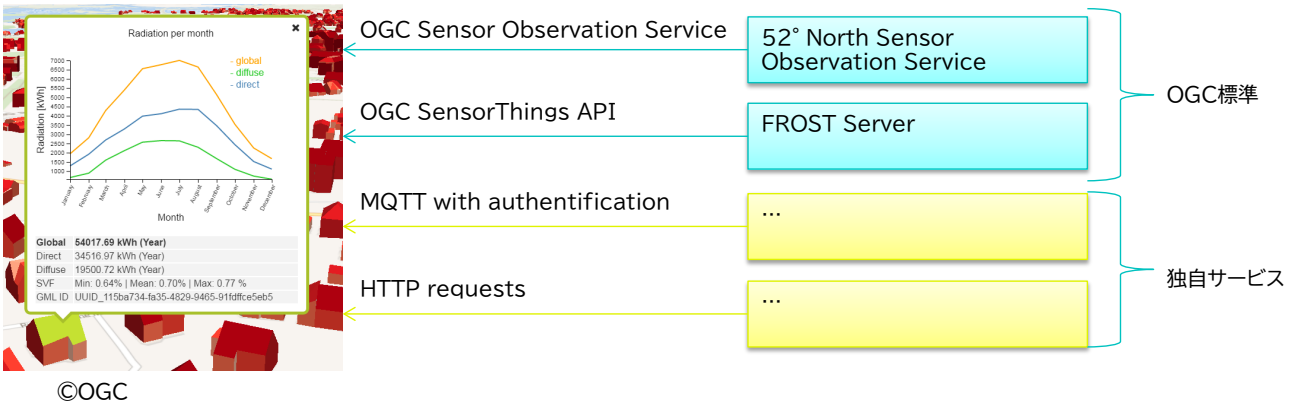



**図 2-22　Dynamizer モジュールの概念モデル（出典[2]一部補足）**

Dynamizer モジュールにおいて中心的な役割を果たす地物は、*Dynamizer* である。全ての都市オブジェクト（*AbstractCityObject*）は、*Dynamizer* をもつことができる（関連役割 *dynamizer*）。*Dynamizer* のステレオタイプが<<FeatureType>>であることからも分かるように、都市モデルにおいて *Dynamizer* のデータが単独で存在することはなく、常に地物のデータの一部として出現する。

*Dynamizer* はセンサーとの接続情報（*SensorConnection*）又は時系列データ（*AbstractTimeseries*）のいずれかをもつ（関連役割 *sensorConnection* 又は *dynamicData*）。また、属性として、取得した動的データの格納先（*attributeRef*）及び動的データを取得する開始時刻（*startTime*）及び終了時刻（*endTime*）をもつ。

*Dynamizer* がセンサーとの接続情報をもつ場合、接続情報を用いてセンサーにアクセスし、値を取得する。接続情報には、必須として参照するセンサーAPI の種類（*connectionType*）及び観測する現象の指定（*observationProperty*）を含まなければならないが、それ以外の属性は、参照するセンサーにより必要なものを記述する。接続対象となるセンサープラットホームは、OGC Sensor Observation Service（SOS）や OGC SensorThings API のような OGC で標準化された Web サービスや API に限定せず、様々なプラットホームを対象としている。




©OGC

**図 2-23　様々なセンサープラットホームと Dynamizer との連携イメージ**

時系列データには、時間位置と値との対が格納されている。時系列データには、単一の時系列データ（*AbstractAtomicTimeseries*）又は複合的な時系列データ（*CompositeTimeseries*）がある。単一の時系列データには、以下の 3 種類がある。

- 表形式の時系列データ（*TabulatedFileTimeseries*）：時系列データを、CSV や XLSX など表形式の外部ファイルとして記述する。
- OGC 標準の時系列データ（*StandardFileTimeseries*）：時系列データを、OGC TimeseriesML などの時系列データや観測データの記述に特化した OGC 標準に従った外部ファイルとして記述する。
- CityGML 形式の時系列データ（*GenericTimeseries*）：時系列データを、CityGML 形式で 3D 都市モデルの中に記述する。

1）及び 2）は、時系列データを外部ファイルとして 3D 都市モデルのファイルとは別に作成する方法であり、3）は 3D 都市モデルの中に直接時系列データを記述する方法である。

*TabulatedFileTimeseries* には、必須の属性として外部ファイルの所在（*fileLocation*）と外部ファイルの形式（*fileType*）が定義され、その他、ヘッダーの行数や区切り文字等、表形式ファイルを読み込むために必要な情報が属性として定義されている。また、*StandardFileTimeseries* にも同様に、必須の属性として外部ファイルの所在（*fileLocation*）と外部ファイルの形式（*fileType*）が定義される。*GenericTimeseries* は、時刻（*timeStamp*）と値との対となる *TimeValuePair* の集まりとして構成され、このとき、値は、整数値、実数値、文字列、幾何オブジェクト、URI、真偽値、*ImplicitGeometry* 及びアピアランスから選択できる。

複合的な時系列データとは、単一の時系列データの組合せで構成される時系列データである。ただし、構成する時系列データに時間の重複は許されない。例えば、1 時間ごとのエネルギー消費量を示す時系列データ（図 2-24）として、平日（月～金）は同じように変化するとした場合、1 週間のエネルギー消費量は、平日 5 回の繰り返しと、土日の集まりとして構成することができる。また、1 週間を示す集まりを繰り返すことで 1 カ月のエネルギー消費量を表現できる。

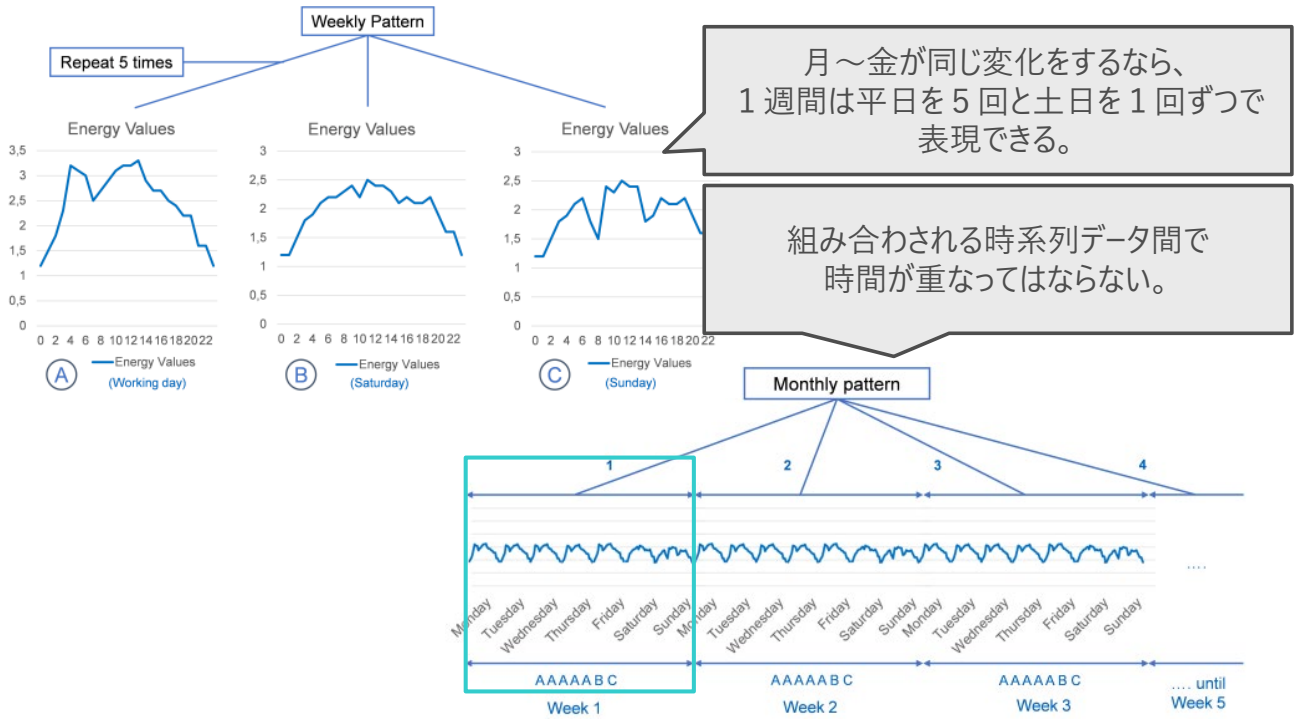



**図 2-24　複合的な時系列データの例（出典[22]一部補足）**

このように、*Dynamizer* はセンサーとの接続情報又は時系列データをもつ。都市オブジェクトは、関連役割 *dynamizer* により、*Dynamizer* をもつことができる。すなわち、建築物、交通オブジェクト、水部、植生など、全ての都市オブジェクトは *Dynamizer* を使って属性の時間的な変化を表現できる。

## 2.8.　Versioning モジュール

Versioning モジュールは、3D 都市モデルの更新情報を記述し、版管理を行うためのモジュールである。

更新情報は、3D 都市モデル単位又は地物単位で記述できる。地物の属性や地物間の関連（例：建築物とその屋根）の単位では変化を記述できないが、地物単位で記述し、更新前後の地物のデータを比較することで、何が変わったかを抽出することはできる。

Versioning モジュールは、*Version*、*VersionTransition* 及び *Transaction* を使用して更新情報を記述する。*Version* には、その版の 3D 都市モデルに含まれる全ての地物（例：*Building*）の ID を記述する。*VersionTransition* には、ある版と別の版を比較し、新旧の 3D 都市モデルに含まれる地物同士の更新情報（*Transaction*）を記述する。

図 2-25 に例を示す。Version“v1”には、建築物の b1 及び b2-1 が含まれている。また、Version“v2”には、建築物の b2-2 及び b3 が含まれている。*VersionTransition* には、v1 と v2 の差分として、3 つの更新情報（*Transaction*）が含まれる。

- v1 の b1 は、v2 では無くなった。
- v1 の b2-1 は、v2 の b2-2 に置き換わった。
- v2 の b3 は、新しく追加された。

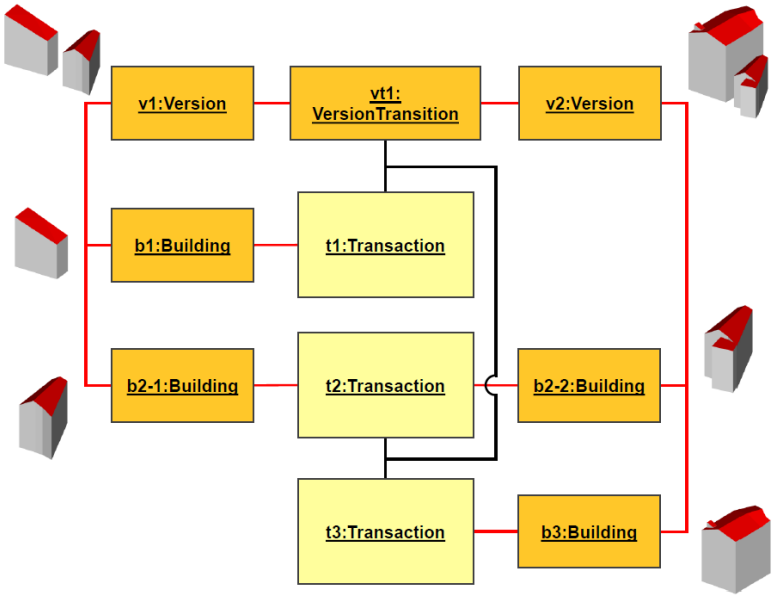



**図 2-25　*Version*、*VersionTransition* 及び *Transaction* の例（出典[23]　©OGC）**

*VersionTransition* は、ある版の都市モデルが別の版の都市モデルに更新された場合に、地物同士の更新情報の記述に使用する。このとき、前後の版の因果関係（更新の理由、版更新の種類）を記述できる。版更新の種類は、*historicalSuccession*、*planned*、*realized*、*fork* 及び *merge* から選択できる。

- *historicalSuccession* は時間的な変化によって版が変わる場合に使用する。
- *planned* 及び *fork* は 1 つの版から複数の後続となる版が作成される際に使用する。
  - *planned* は都市の将来予測となる版に適用し、*fork* はそれ以外の理由により複数の版が作成される場合に適用する。
- *realized* 及び *merge* は異なる版が、1 つの版に統合される場合に使用する。
  - *realized* は計画された複数の版から選択された 1 つの版に適用し、*merge* はそれ以外の理由により統合される場合に使用する。

なお、*VersionTransition* には、1 つ以上の地物同士の更新情報（*Transaction*）が含まれなければならない。

*Transaction* には地物の更新情報を記述する。二つの版を比較し、各々に含まれる地物（*oldFeature*、*newFeature*）と更新の内容（*type*）を記述する。*oldFeature* には、古い版に含まれる地物の ID を記述する。*newFeature* には、これに対応する新しい版に含まれる地物の ID を記述する。

*type* は、更新の内容を *delete*、*replace* 又は *insert* から選択して記述する。

- *delete* は削除される場合であり、*oldFeature* のみを記述し、*newFeature* は記述しない。
- *replace* は置換される場合であり、*oldFeature* と *newFeature* との両方を記述する。
- *insert* は挿入される場合であり、*oldFeature* は空（対応する地物がない）となり、*newFeature* のみを記述する。

図 2-26 に例を示す。b1 は、削除されたため、*Transaction* t1 において、*oldFeature* として指定され、*newFeature* は記述されない。b2-1 は、b2-2 に置き換わったため、*Transaction* t2 では、*oldFeature* として b2-1 が指定され、*newFeature* には b2-2 が指定された。b3 は挿入されたため、*Transaction* t3 では、*oldFeature* は指定されず、*newFeature* には b3 が指定された。

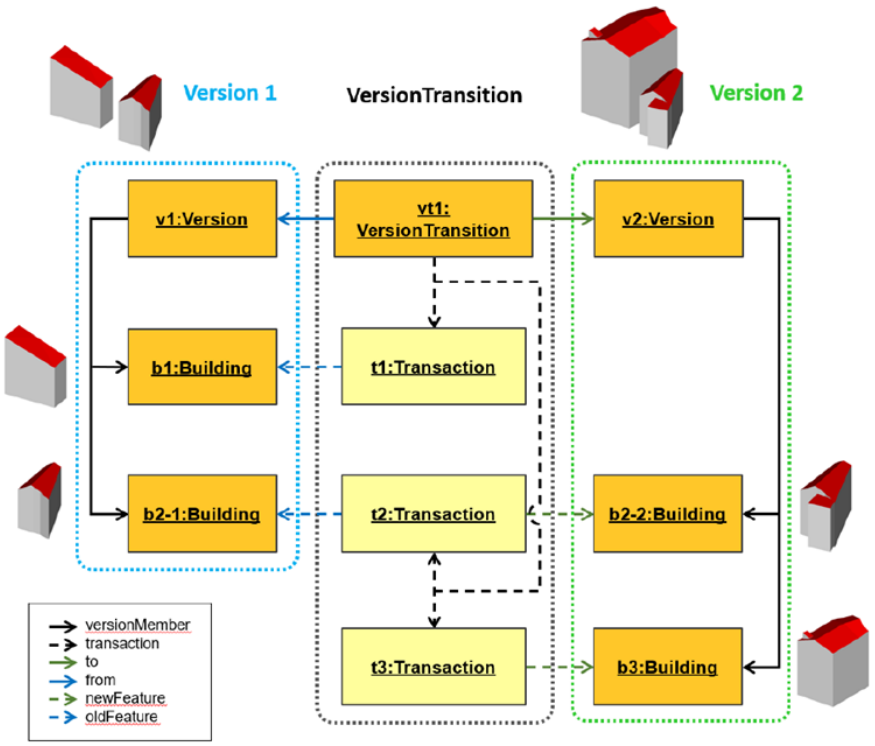



**図 2-26　*Transaction* の例（出典[23]　©OGC）**

Versioning モジュールの概念モデルを図 2-27 図 2-272-28 に示す。

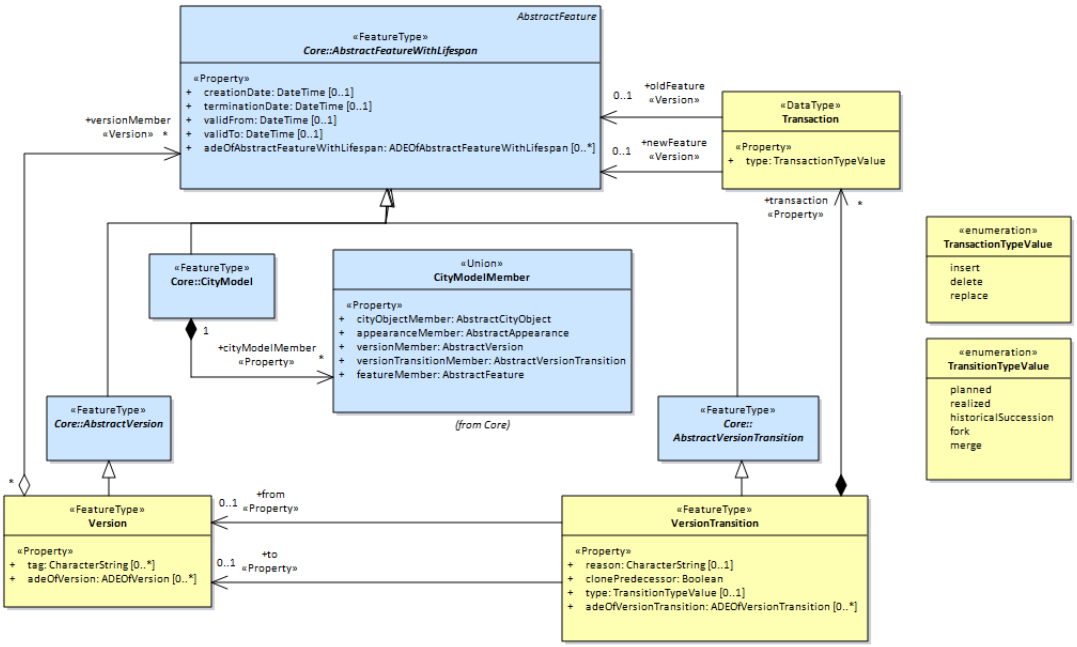



**図 2-27　Versioning モジュールの概念モデル（出典[2] ©OGC）**

都市モデル（*CityModel*）は、構成要素（*cityModelMember*）として版情報をもつことができる（*versionMember* 及び

*versionTransitionMember*)。

*Version* は、都市モデルの版を定義し、属性 *tag* を用いて任意のラベルを付けることができる。*Version* は、その版に含める地物を参照できる（関連役割 *versionMember*)。

*VersionTransition* は、Version への関連役割 *from* 及び *to* により、都市モデル単位の版管理を行うことができる。前の版は *from*、後の版は *to* により指定する。*VersionTransition* は、*Transaction* によって、地物単位の版管理を行うことができる。関連役割 *oldFeature* により更新前、*newFeature* により更新後の地物を指定する。*VersionTransition* には版更新の理由（*reason*）及び版更新の種類（*type*）の他、含むべき地物の対象（*clonePredecessor*）を指定する。これは、後継となる版の都市モデルに含まれるべき地物を取得する方法を示す属性であり、真偽値（*true* 又は *false*）で示す（図 2-28)。値が *false* の場合は、地物集合が後継となる版の中で明示的に列挙されることを意味し、つまり、前の版と後の版との地物を全て記述する。一方値が *true* の場合は前の版の都市モデルと *Transaction* に示された変更箇所のリストから、後継となる版の都市モデルを派生させなければならないことを意味し、つまり、前の版と変更点のみが記述されるので、そこから後継となる都市モデルに含まれるべき地物をデータの利用者が導出しなければならない。


**図 2-28　*clonePredecessor* の例**

## 2.9.　PointCloud モジュール

PointCloud モジュールは、CityGML 3.0 で新たに追加されたモジュールであり、地物の幾何形状（物理的な空間又は面）に対応する点群への参照を定義する。PointCloud モジュールを使用することにより、個々の地物またはその集まりから点群を参照できるようになる。PointCloud モジュールは、物理的な空間やその境界面の形状を点群のみで記述することを許容している。つまり、地物の形状として、幾何オブジェクトと点群データの両方を用いて記述してもよいし、幾何オブジェクトを作成せず、点群だけでもよい。点群の記述方法には、大きく 2 種類がある。

- *MultiPoint* を使用し、CityGML 内にインラインで記述する。
- LAS や LAZ 等で記述された外部の点群データファイルを参照する。

LAS（LASer）形式は、3 次元点群データを交換するためのバイナリファイル形式として、ASPRS (American Society for Photogrammetry and Remote Sensing) によって開発されたフォーマットであり、現在では OGC 標準

（https://www.ogc.org/standard/las/）となっている。LAZ 形式は LAS ファイルを圧縮した形式である。この仕様はドキュメントとしては公開されていないが、LAZ 形式への変換にはオープンソースのライブラリである LASZip を使用することができる。

外部の点群データファイルを参照する場合、外部ファイルを指定する方法と、外部ファイルの一部を指定する方法とがある。

図 2-29 に PointCloud モジュールの概念モデルを示す。物理的な空間を示す *AbstractPhysicalSpace* 及び地物の境界となる *AbstractThematicSurface* は、点群である *PointCloud* を参照できる（関連役割 *pointCloud*）。*PointCloud* には、点群の形式を示す属性（*mimeType*）の他、外部ファイルで記述されている場合にファイルを指定する *pointFile*、また、点群ファイルに適用された空間参照系を特定するための *pointFileSrsName* を属性としてもつ。点群を外部ファイルではなくインラインで記述する場合には、関連役割 *points* により、*MultiPoint*（点の集まり）を使用して記述する。

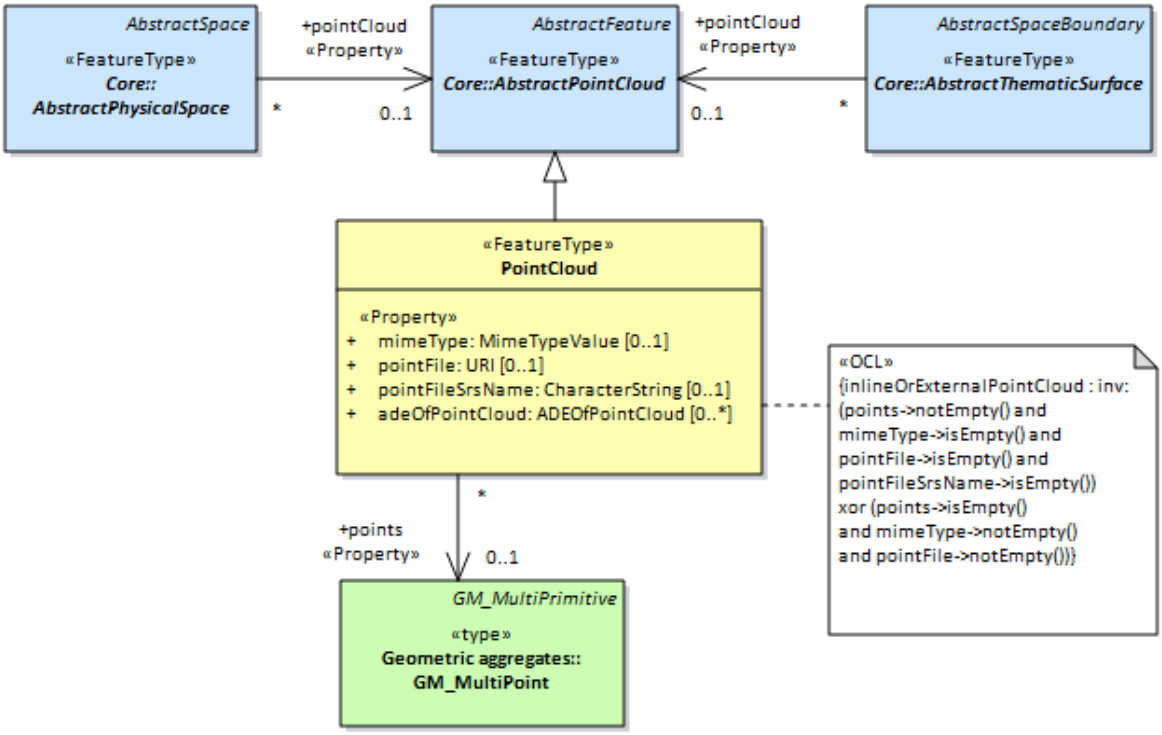



**図 2-29　PointCloud モジュールの概念モデル（出典[2] ©OGC）**

# 3. CityGML 3.0 GML による実装

## 3.1. CityGML 3.0 における符号化仕様

CityGML 3.0 では、モデル駆動型アプローチ（Model-driven approach）が採用され、ISO 19100 シリーズに準拠した UML による概念モデルと、それから自動的に生成された符号化仕様から構成される。CityGML 2.0 では概念モデルの記述に符号化仕様が混在[3]していたが、CityGML 3.0 では概念モデルと符号化仕様と完全に分離され、標準仕様としても、「OGC City Geography Markup Language (CityGML) Part 1: Conceptual Model Standard」及び「OGC City Geography Markup Language (CityGML) 3.0 Part 2: GML Encoding Standard」として分けて発行されることとなった。また、CityGML 2.0 では、参照する GML 版が 3.1.1 が参照されていたのに対し、CityGML 3.0 では、GML 3.2.2 が参照される。この GML 3.2.2 は、ISO 19136 Geography Markup Language と一致する。すなわち、CityGML 3.0 Part 2 は、ISO 19136 に準拠している。

CityGML SWG は、少なくとも Part 1 及び Part 2 の 2 つの仕様を発行するとしており、今後は Part3 として GML 以外の符号化仕様が作成される可能性に言及している（図 3-1）。ただし、CityGML 3.0 概念モデルに対応した符号化仕様とするには、CityGML 3.0 Part1 に示された適合性試験に合格し、概念モデルに準拠していることを示さなければならない。

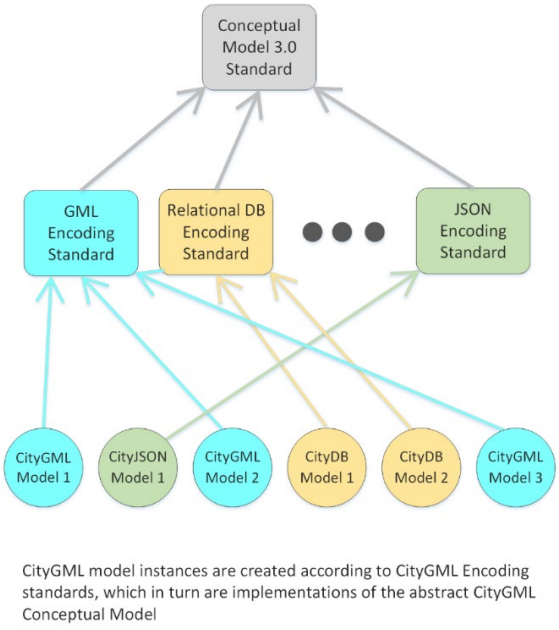



**図 3-1 CityGML 3.0 における交換フォーマットの考え方（出典[16] ©OGC）**

実際に、JSON ベースの CityJSON が開発され、CityJSON v1.0 は OGC のコミュニティ標準として承認されている。CityJSON は CityGML 2.0 に定義された概念モデルを網羅してはおらず、CityGML 2.0 のサブセット[4]という位置づけである。一方で、QGIS で読み込めるプラグイン（CityJSON Loader）が開発されたことで 3D 都市モデルの利用可能性を拡げることができている（[8]）。CityJSON は引き続き CityGML 3.0 に対応する版の検討が現在進められている。

図 3-1 の CityDB は、CityGML を実装するリレーショナルデータベース（RDB）を指す。CityGML に適合する RDB として、ミ

[3] CityGML 2.0 の概念モデルでは、属性の型として xs:string や gml:CodeType が使用されるなど、GML による符号化が前提のモデルとなっていた。

[4] CityJSON v1.0 と CityGML 2.0 の適合性（https://www.cityjson.org/citygml/v20/）

ュンヘン工科大学やvirtualcitySYSTEMS社がその開発に参画しているオープンソースの3DCityDB(https://www.3dcitydb.org/)があるが、現状ではOGC内で標準策定に向けた動きはない。他方3Dモデルやシーンを表現するフォーマットであるglTFについては、3Dグラフィックスや拡張現実等に関する標準化団体であるKhronos Group Inc.との協働により、CityGML 3.0の符号化に向けたPoC（Proof of Concept：概念実証）が行われている。

## 3.2. CityGML 3.0 の XML Schema

### 3.2.1. XML Schema の構造

CityGML 3.0では、概念モデルに対応するXML Schemaが作成される。XML SchemaはUMLクラス図に記述されたクラス、属性及び関連役割がタグとして定義され、UMLクラス図と1対1に対応したXML Schemaが作成される。

CityGML 3.0のXML Schemaは、モジュールごとに作成される（表 3-1）。

**表 3-1 CityGML 3.0 の XML Schema ファイル**

| モジュール | 接頭辞 | 名前空間 | ファイル名 |
|---|---|---|---|
| Core | core | http://www.opengis.net/citygml/3.0 | core.xsd |
| Appearance | app | http://www.opengis.net/citygml/appearance/3.0 | appearance.xsd |
| Generics | gen | http://www.opengis.net/citygml/generics/3.0 | generics.xsd |
| Dynamizer | dyn | http://www.opengis.net/citygml/dynamizer/3.0 | dynamizer.xsd |
| Versioning | vers | http://www.opengis.net/citygml/versioning/3.0 | versioning.xsd |
| PointCloud | pcl | http://www.opengis.net/citygml/pointcloud/3.0 | pointCloud.xsd |
| Construction | con | http://www.opengis.net/citygml/construction/3.0 | construction.xsd |
| Building | bldg | http://www.opengis.net/citygml/building/3.0 | building.xsd |
| Bridge | brid | http://www.opengis.net/citygml/bridge/3.0 | bridge.xsd |
| Tunnel | tun | http://www.opengis.net/citygml/tunnel/3.0 | tunnel.xsd |
| CityFurniture | frn | http://www.opengis.net/citygml/cityfurniture/3.0 | cityFurniture.xsd |
| CityObjectGroup | grp | http://www.opengis.net/citygml/cityobjectgroup/3.0 | cityObjectGroup.xsd |
| LandUse | luse | http://www.opengis.net/citygml/landuse/3.0 | landUse.xsd |
| Relief | dem | http://www.opengis.net/citygml/relief/3.0 | relief.xsd |
| Transportation | tran | http://www.opengis.net/citygml/transportation/3.0 | transportation.xsd |
| Vegetation | veg | http://www.opengis.net/citygml/vegetation/3.0 | vegetation.xsd |
| WaterBody | wtr | http://www.opengis.net/citygml/waterbody/3.0 | waterBody.xsd |

また、CityGML 3.0では、XML Schemaの可読性を向上させるためのSchema Documentation[5]が作成された。表 3-1に示すXML Schemaごとに、名前空間やインポートする他のXML Schemaや接頭辞の宣言などの概要に加え、要素ごとにXML Schemaにおけるタグの階層構造の図やインスタンスの記述例が示されており、視覚的にXML Schemaの構造を理解しやすい（図 3-2）。

[5] https://opengeospatial.github.io/CityGML-3.0Encodings/xsd-doc/3.0/ [2023/06/30 アクセス]

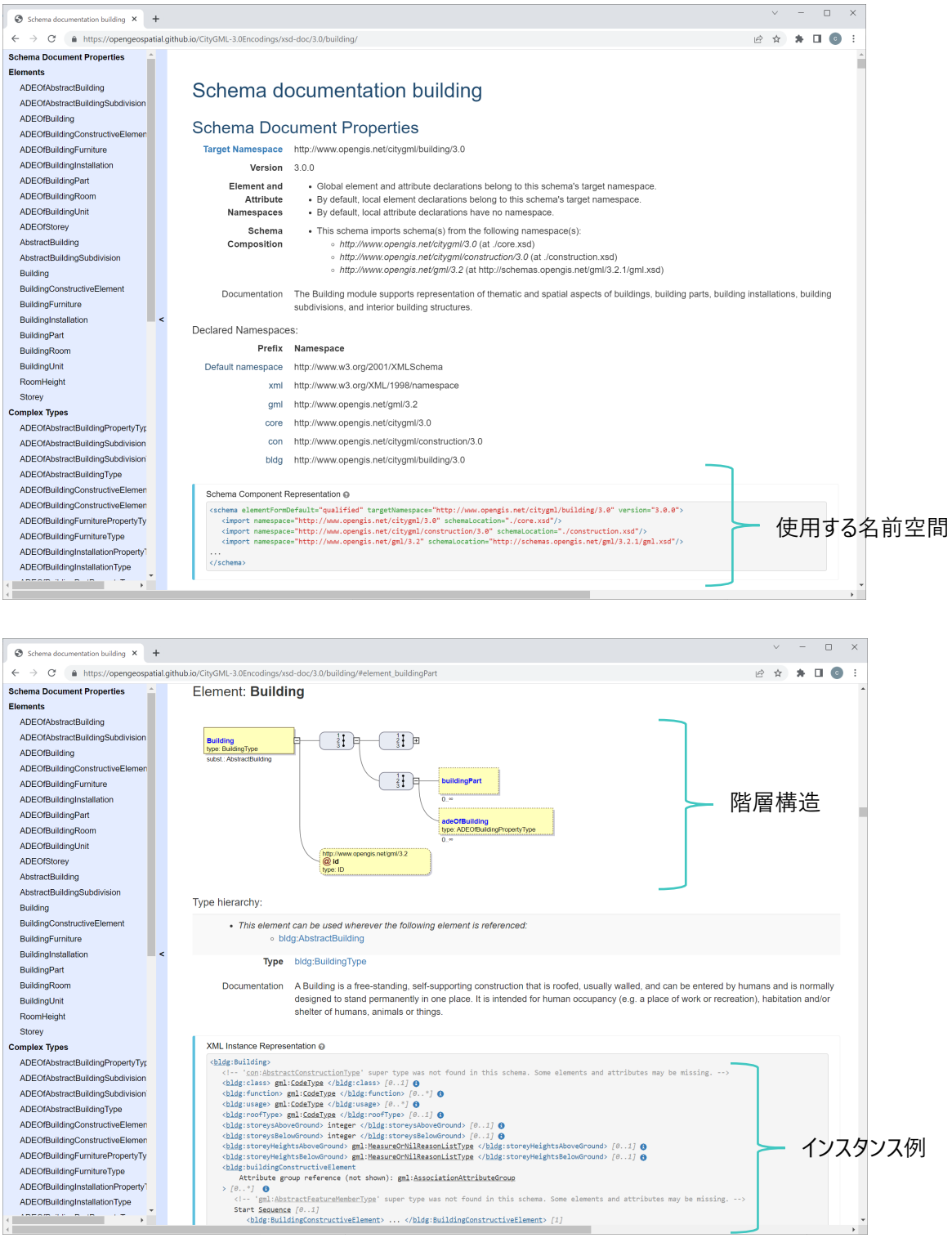




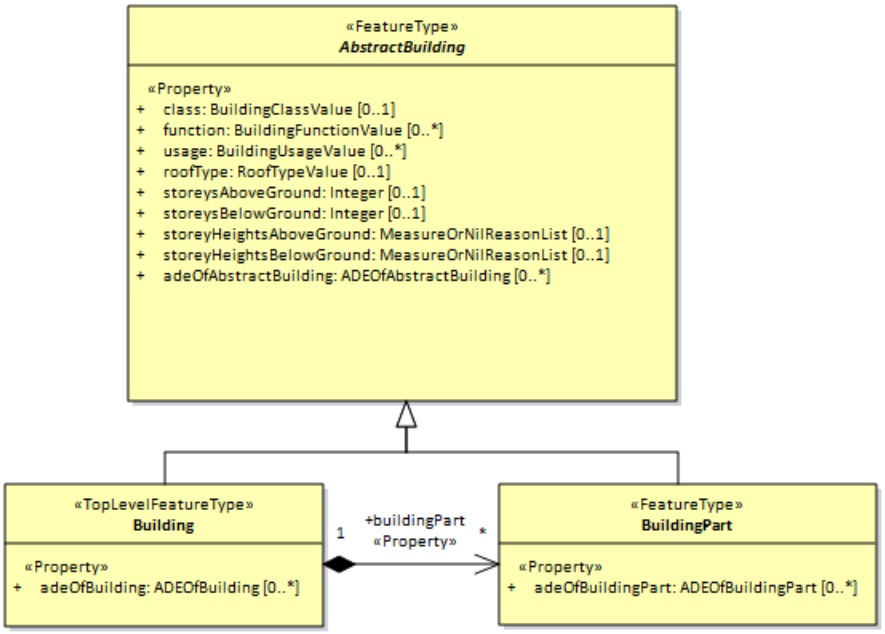


**図 3-2　Schema Documentation の例**

以下で、概念モデルと GML による符号化仕様の対応を概説する。XML Schema では、概念モデルにクラスとして記述された地物等に対応する要素が宣言される。UML クラス図から XML Schema への符号化規則は、ISO 19136 Geography Markup Language

の Annex E UML-to-GML application schema encoding rules[6]に詳細が示されている。

例として図 3-3 に建築物（*Building*）の概念モデルと XML Schema を示す。




```
<element name="Building" substitutionGroup="bldg:AbstractBuilding" type="bldg:BuildingType"/>

<complexType name="BuildingType">
  <complexContent>
    <extension base="bldg:AbstractBuildingType">
      <sequence>
        <element maxOccurs="unbounded" minOccurs="0" name="buildingPart">
          <complexType>
            <complexContent>
              <extension base="gml:AbstractFeatureMemberType">
                <sequence minOccurs="0">
                  <element ref="bldg:BuildingPart"/>
                </sequence>
                <attributeGroup ref="gml:AssociationAttributeGroup"/>
              </extension>
            </complexContent>
          </complexType>
        </element>
        <element maxOccurs="unbounded" minOccurs="0" name="adeOfBuilding" type="bldg:ADEOfBuildingPropertyType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="BuildingPropertyType">
  <sequence minOccurs="0">
    <element ref="bldg:Building"/>
  </sequence>
  <attributeGroup ref="gml:AssociationAttributeGroup"/>
  <attributeGroup ref="gml:OwnershipAttributeGroup"/>
</complexType>
```

属性及び関連役割の要素宣言

要素宣言

複合型宣言

他の地物から
参照用の
複合型宣言

**図 3-3　*Building* の UML クラス図と XML Schema**

地物（概念モデルでステレオタイプが<<TopLevelFeatureType>>又は<<FeatureType>>）は、XML Schema において、地物の名称を使った要素宣言が行われる。これにより、XML では地物の名称がタグとして出現できる。また、地物の名称の開始タグと

[6] ISO 19136 は、和訳され JIS X7136 として発行されている。

終了タグの間に含むべき内容を示す複合型宣言が行われる。この複合型宣言には、地物に定義された属性及び関連役割の要素宣言が含まれる。これにより、地物の開始タグと終了タグの間に地物の属性や関連役割のタグが出現できる。要素宣言は当該地物の上位概念となる地物の代替として使用可能（substitutionGroup）であること、また、複合型宣言は当該地物の上位概念となる地物を拡張（extension）して定義される。

さらに、この地物が他の地物から参照される場合に備え、参照用の複合型宣言が行われる。参照用の複合型宣言ではその名称として XXXPropertyType（XXX にはクラスの名称が入る）が指定される。参照用の複合型宣言は、XLink による ID 参照を可能とする仕組みである。

*Building* の場合、地物の名称である"Building"をタグとして使用するために、要素宣言（name="Building"）を行う。このとき、"Building"のタグが *Building* の上位の地物である *AbstractBuilding* の代替として出現できるよう指定する（substitutionGroup="bldg:AbstractBuilding"）。

Building の開始タグ及び終了タグの間に含むべき内容は、複合型宣言において記述する。複合型の名称は、XXXType（XXX にはクラスの名称が入る）とし、*Building* の場合は BuildingType となる。複合型は、*Building* の上位の地物である *AbstractBuilding* の複合型（AbstractBuildingType）を拡張する（extension="bldg:AbstractBuildingType"）。*Building* には、*BuildingPart* との関連を示す関連役割 *buildingPart* 及び拡張のための属性 *adeOfBuilding* が定義されているため、各々に対応する要素宣言が、BuildingType の複合型宣言で行われる。属性及び関連役割の多重度は、minOccurs 及び maxOccurs を用いて指定する。

また、概念モデルで集成（白抜きの菱形）又は合成（黒塗りの菱形）として表現されている場合、その関連役割の要素は、GML に定義された複合型である gml:AbstractFeatureMemberType を拡張して定義する。一方、概念モデルで関連（菱形がつかない矢印）又は属性の型が他のクラスとして定義されている場合、その要素宣言では、要素の型として参照用の複合型を指定する。

*Building* の場合、*BuildingPart* への関連は合成として概念モデルで記述されている。そこで、buildingPart の要素宣言は、gml:AbstractFeatureMemberType を拡張して定義する。一方、属性 *adeOfBuilding* の型である *ADEOfBuilding* は概念モデルでクラスとして定義されているため、adeOfBuilding の要素宣言では、参照用の複合型である ADEOfBuildingPropertyType を指定する。

複合型宣言では、属性及び関連役割の要素宣言は、順序が付けられている（<sequence>）。これは宣言された順序でしか XML には出現することができないことを意味する。建築物の例では、複合型宣言の中で buildingPart が最初に宣言され、その次に adeOfBuilding が宣言されている。よって、XML では、buildingPart のタグは adeOfBuilding のタグよりも前に出現する（ただし、buildingPart 及び adeOfBuilding はいずれも多重度が 0 以上となっており、省略可能である。XML には、buildingPart のみ又は adeOfBuilding のみが出現する場合もある）。

なお、CityGML 3.0 では、XML Schema は UML クラス図を使って記述された概念モデルから自動的に出力される。自動化により CityGML 全体での論理的な整合性を確保できる。自動出力は標準として定義されたモジュールだけではなく、ADE も対象となる。UML クラス図からの XML Schema の生成には、ShapeChange（https://shapechange.net/）が使用されている。ShapeChange はモデル駆動型アプローチの一環として、CityGML だけではなく、今後、OGC で作成される他の標準においても概念モデルから符号化仕様を作成する場合に使用される予定である。

### 3.2.2. XML データの構造

CityGML 形式の 3D 都市モデルを作成する場合は、CityGML 2.0 と同様に 3D 都市モデルに含める地物が定義された XML Schema の所在（schemaLocation）を指定する（図 3-4）。また、指定した XML Schema 文書に対する接頭辞は、表 3-1 に示す接頭辞を採用することが原則となる。

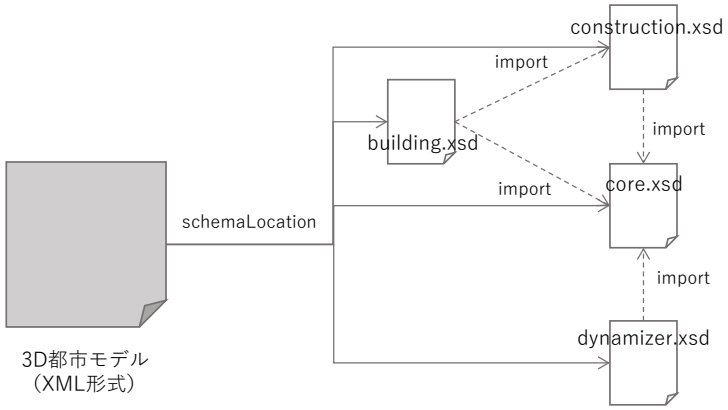



**図 3-4　schemaLocation の指定の例**

3D 都市モデルでは、指定した XML Schema に定義された出現順序、入れ子構造、回数で地物及びその属性や関連役割がタグとして出現する。このとき、拡張（**extension**）により追加されたタグは後ろに追加されていくため、より抽象度の高い要素（概念モデルで上位に定義された地物）に定義されたタグから出現する。

例えば、*Building* は以下に示す地物を継承し、定義されている。

*AbstractFeature* > *AbstractFeatureWithLifespan* > *AbstractCityObject* > *AbstractSpace* > *AbstractPhysicalSpace* > *AbstractOccupiedSpace* > *AbstractConstruction* > *AbstractBuilding*

この場合、XML Schema でも複合型である **AbstractFeatureType** を拡張し **AbstractFeatureWithLifespanType** が定義され、**AbstractFeatureWithLifespanType** を拡張し **AbstractCityObjectType** が定義されるというようにして、各々の地物に定義された属性や関連役割がタグとして複合型宣言で追加される。よって、**Building** の XML には、最初に *AbstractFeature* に定義された属性や関連役割に対応するタグ、次に *AbstractFeatureWithLifespan* に定義された属性や関連役割に対応するタグ、さらに次に *AbstractCityObject* に定義された属性や関連役割に対応するタグという順序になる。

図 3-5 に、建築物の形状を LOD2 の立体として表現し、付属物を区分し、さらに *Dynamizer* によるセンサー情報を付与した場合のタグの出現順序を例示する。都市モデルを示す **CityModel** の子要素（**cityObjectMember**）として **Building** を記述する。**Building** の子要素として最初に **dynamizer**（*Dynamizer* への参照）が出現する。**dynamizer** は **AbstractCityObjectType** において定義されているタグである。次に、**AbstractSpaceType** に定義されたタグである LOD2 の立体（**lod2Solid**）が出現する。最後に **AbstractBuildingType** に定義されたタグである付属物への参照（**buildingInstallation**）が出現する。

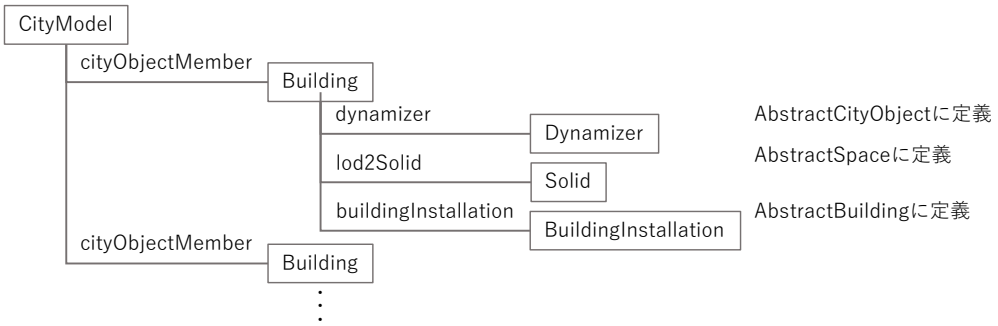



**図 3-5　タグの出現順序の例**

CityGML 3.0 では、共通する属性や関連役割を上位概念に定義することになったため、継承する属性や関連役割に対応するタグの出現場所や出現順序が CityGML 2.0 の場合と異なることに注意する必要がある。

### 3.2.3. CityGML 3.0 GML への対応状況

CityGML 3.0 Part 2 GML Encoding Standard は 2023 年 6 月末に発行されたばかりであり、ソフトウェアなどの対応も今後順次進んでいくと想定される。

CityGML 3.0 Part 2 に対応したテストデータの実装は、CityGML 3.0 の仕様検証のために開催された OGC CityGML 3.0 Hackathon in June 2019 in London（https://members.geovation.uk/ca/events/view/1036784763/ogc-citygml-hackathon）及び CityGML Challenge in October 2019 in Manchester（https://www.ogc.org/initiatives/citygmlchalleng/）で行われた。CityGML 3.0 のサンプルデータは、CityGML SWG が管理する技術資料のポータルサイト[4]より入手可能である。CityGML Challenge では CityGML 3.0 のドラフト（当時）に従ったデータに対応するアプリケーションが開発され、virtualcitySYSTEMS 社による Manchester 3D Data Viewer（図 3-6）が賞を獲得している。


**図 3-6 Manchester 3D Data Viewer[7] ©OGC**

CityGML の読み込みを行うオープンソースの Java API である citygml4j（https://github.com/citygml4j/citygml4j）は CityGML 3.0 に対応している。また、CityGML 2.0 から CityGML 3.0 に変換するツール citygml2-to-citygml3（https://github.com/tum-gis/citygml2-to-citygml3）や、IFC のデータセットを CityGML 3.0 に変換する FME のワークスペース ifc-to-citygml3（https://github.com/tum-gis/ifc-to-citygml3）が公開されている。

なお、CityGML 3.0 は GML 3.2 に準拠した応用スキーマであることから、GML 3.2 をサポートするツール（FME、FZKViewer、GDAL[8]等）であれば CityGML 3.0 にも対応できる。2023 年 6 月時点で CityGML 3.0 形式で 3D 都市モデルを整備している事例は把握できていない。しかしながら INSPIRE のように ISO 標準を作業の基礎と位置付けている場合は、CityGML 3.0 Part 2 において採用される GML 3.2 が ISO 19136 に準拠となることから、CityGML 2.0 を CityGML 3.0 に変換して利用する事例が今後増えていくと想定される。

---

[7] https://www.ogc.org/press-release/ogc-announces-virtualcitysystems-as-the-winner-of-the-2019-ogc-citygml-challenge/

[8] GDAL(Geospatial Data Abstraction Library）は、OSGeo 財団が X/MIT ライセンスにより提供している、ラスターおよびベクター地理空間情報データフォーマットのための変換用ライブラリ

## 3.3. Construction モジュール

### 3.3.1. Construction モジュールの実装方法

Construction モジュールの実装には、*OtherConstruction* を使用する。*OtherConstruction* は *AbstractConstruction* から継承する属性や関連役割をタグとして使用できる。よって、*OtherConstruction* も、*WallSurface*、*GroundSurface*、*RoofSurface* などの境界面や、境界面上に存在する *Door* や *Window* を構造物の部品として記述できる。これは、建築物、橋梁及びトンネルと同様のデータ構造である。

しかし *AbstractConstruction* は、その部品として、構造部材（*AbstractConstructiveElement*）、付属物（*AbstractInstallation*）及び家具（*AbstractFurniture*）を記述できるデータ構造にはなっていない。これにより、*AbstractConstruction* を継承する *OtherConstruction* も部品として構造部材、付属物及び家具を記述できない。そのため、*OtherConstruction* の部品として構造部材や付属物等を記述したい場合は、ADE による拡張が必要である。

例えば、*OtherConstruction* に構造部材を記述したい場合、以下の手順で概念モデルを拡張する（図 3-7）。

①*OtherConstruction* の構造部材を表す地物を、上位の概念となる *AbstractConstructiveElement* を継承して定義する。
②*OtherConstruction* が追加した地物（*OtherConstructiveElement*）をもてるよう、関連役割を追加する。

*OtherConstruction* に関連役割を追加する場合は、*ADEOfOtherConstruction* を拡張したデータ型（*OtherConstructionProperties*）を作成し、これに関連役割（*otherConstructiveElement*）を追加する。*ADEOfOtherConstruction* は、*OtherConstruction* に属性や関連役割を追加する場合に備えてあらかじめ CityGML 3.0 に用意された拡張用のデータ型である。

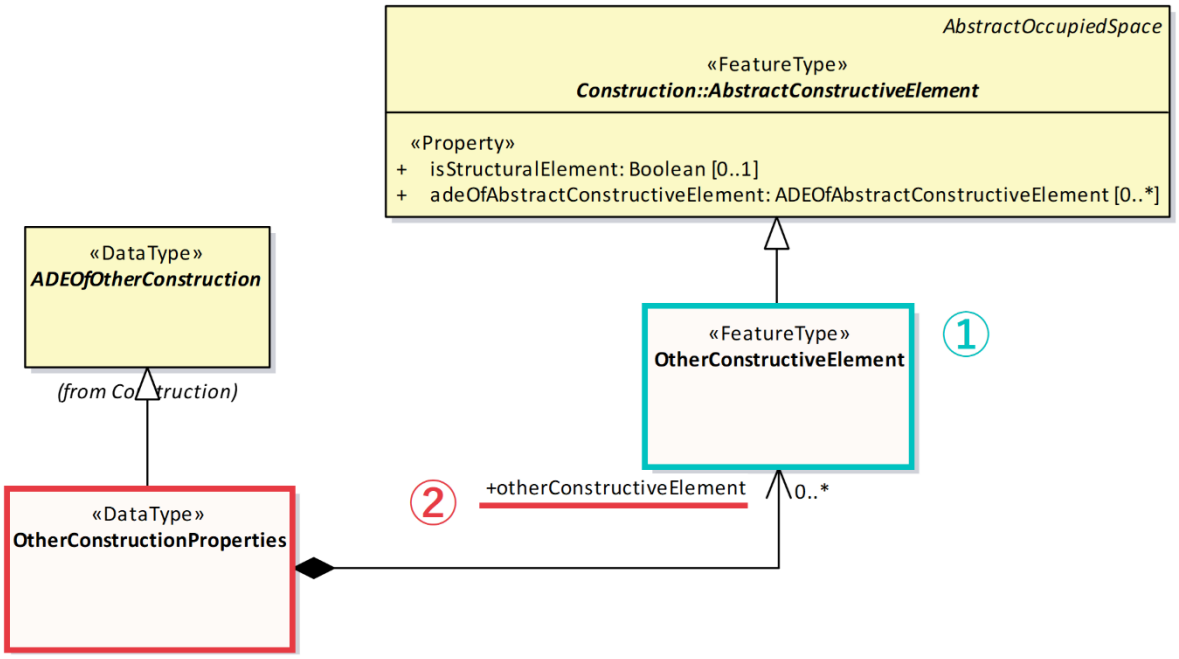



**図 3-7　OtherConstruction の拡張例**

この概念モデルに対応する XML Schema は以下のようになる。

- 地物型 OtherConstructiveElement の XML Schema

*OtherConstruction* の構造部材を表す地物を、上位の概念となる *AbstractConstructiveElement* を継承して定義するため、構造部材（*OtherConstructiveElement*）に対応する要素宣言において con:AbstractConstructiveElement の代替グループ（substitutionGroup）であることを指定し、複合型宣言では con:AbstractConstructiveElementType を拡張（extension）する。

```
<element name="OtherConstructiveElement" type="uro:OtherConstructiveElementType"
substitutionGroup="con:AbstractConstructiveElement" /> <!-- substitutionGroup を指定 -->
 <complexType abstract="true" name="OtherConstructiveElementType">
    <complexContent>
      <extension base="con:AbstractConstructiveElementType"/><!-- extension を指定 -->
    </complexContent>
 </complexType>
  <complexType name="OtherConstructiveElementPropertyType">
    <sequence minOccurs="0">
      <element ref="uro:OtherConstructiveElement"/>
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup"/>
    <attributeGroup ref="gml:OwnershipAttributeGroup"/>
 </complexType>
```

- OtherConstruction の拡張プロパティの XML Schema

*OtherConstruction* が追加した地物型 *OtherConstructiveElement* を部品としてもつことができるよう、関連役割を追加する。拡張したデータ型（*OtherConstructionProperties*）に対応する要素宣言において con:ADEOfOtherConstruction の代替グループ（substitutionGroup）であることを指定し、複合型宣言では con:ADEOfOtherConstructionType を拡張（extension）する。

この複合型宣言において、関連役割 *otherConstructiveElement* に対応する要素を宣言する。

```
   <element          name="OtherConstructionProperties"          substitutionGroup="con:ADEOfOtherConstruction"
type="uro:OtherConstructionPropertiesType"/><!-- substitutionGroup を指定 -->
   <complexType name="OtherConstructionPropertiesType">
     <complexContent>
       <extension base="con:ADEOfOtherConstructionType"> <!-- extension を指定 -->
         <sequence>
           <element maxOccurs="unbounded" minOccurs="0" name="otherConstructiveElement">
              <!-- 関連役割の要素を宣言 -->
             <complexType>
               <complexContent>
                 <extension base="gml:AbstractFeatureMemberType">
                   <sequence minOccurs="0">
                     <element ref="uro:OtherConstructiveElement"/>
                   </sequence>
                   <attributeGroup ref="gml:AssociationAttributeGroup"/>
                 </extension>
               </complexContent>
             </complexType>
           </element>
```

```
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="OtherConstructionPropertiesPropertyType">
    <sequence>
      <element ref="uro:OtherConstructionProperties"/>
    </sequence>
</complexType>
```

### 3.3.2. Construction モジュールの導入に向けた課題

Construction モジュールを使用することにより、堤防や防波堤など都市に存在する橋梁やトンネル以外の構造物を記述できるようになる。施設管理や災害シミュレーションなど 3D 都市モデルに統合するニーズも高いと考えられる。Construction モジュールのデータ構造は、橋梁やトンネルと同様であるため、データ作成上大きな問題はない。しかし、CityGML 3.0 では *OtherConstruction* の部品として構造部材、付属物及び家具を記述するための要素が宣言されておらず、これらの記述には ADE が必要となる。データ整備主体毎に ADE を開発することは似て非なるデータ構造の 3D 都市モデルが作成されることになり、共通利用可能な標準的な ADE を用意することが必要である。

また、ダムや河川構造物に関しては、BIM/CIM の導入が進められている。設計や施工段階又は維持管理の段階で作成されたデータを活用し、3D 都市モデルを作成するという方法も考えられる。そのためには、BIM/CIM モデルと Construction モジュールとの標準的な対応付けを定める必要がある。

## 3.4. Dynamizer モジュール

### 3.4.1. Dynamizer モジュールの実装方法

Dynamizer は、センサーとの接続情報又は時系列データのいずれかをもつ（2.7 参照）が、時系列データを外部ファイルで記述する場合を除き、GML 形式で 3D 都市モデルと一体的に符号化できる。

例えば、図 3-8 は、建築物（**bldg:Building**）の部屋（**bldg:BuildingRoom**）に設置された温度センサーの値を入力データとして、部屋がもつ属性に反映させたい場合の記述である。**dyn:Dynamizer** には、入力元となるセンサーデータへの接続情報（**dyn:SensorConnection**）が記述され、出力先となる地物の属性値を **dyn:attributeRef** の中で指定する。また、この例の場合には、データを取得する時間範囲を指定している。

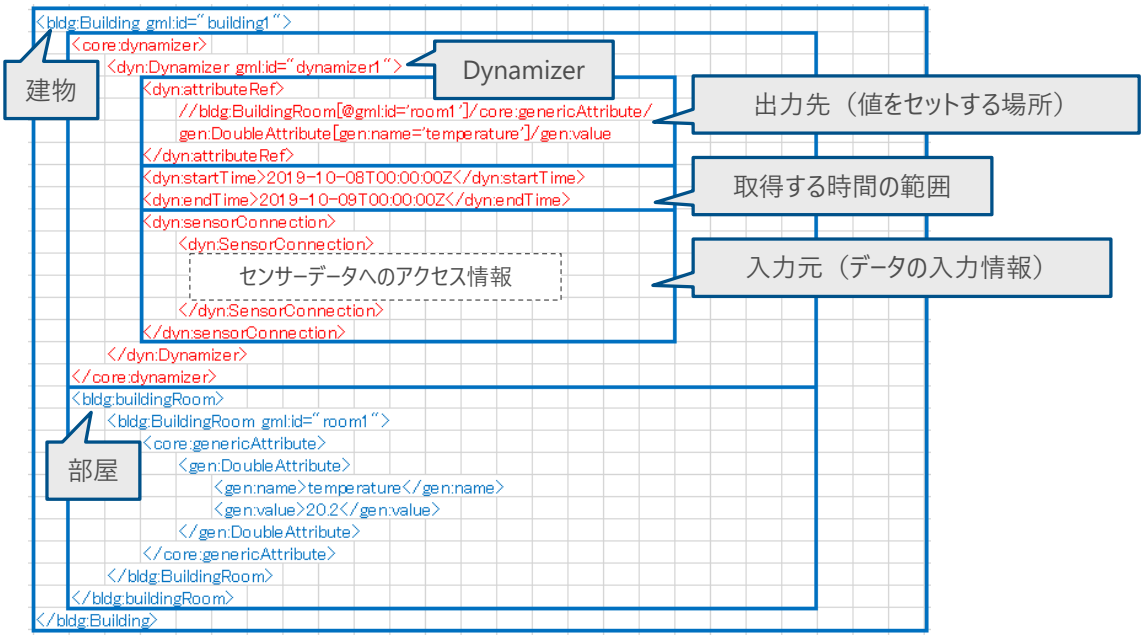



**図 3-8　Dynamizer の実装イメージ**

出力先の指定の記法には、XPath（XML Path Language）を使用する。XPath は、XML 文書の特定の部分を指し示す構文を規定する記法である。指定したい部分のタグを、入れ子構造の外側から記載し、[ ]内に範囲を絞り込む条件を記述する。この例の場合は、まず、bldg:BuildingRoom ＞ gen:genericAttribute ＞ gen:DoubleAttribute ＞ gen:value と出力先のタグを指定し、かつ、bldg:BuildingRoom は gml:id='room1'、gen:DoubleAttribute は name='temperature'と絞り込みの条件を設けている（図 3-9）。

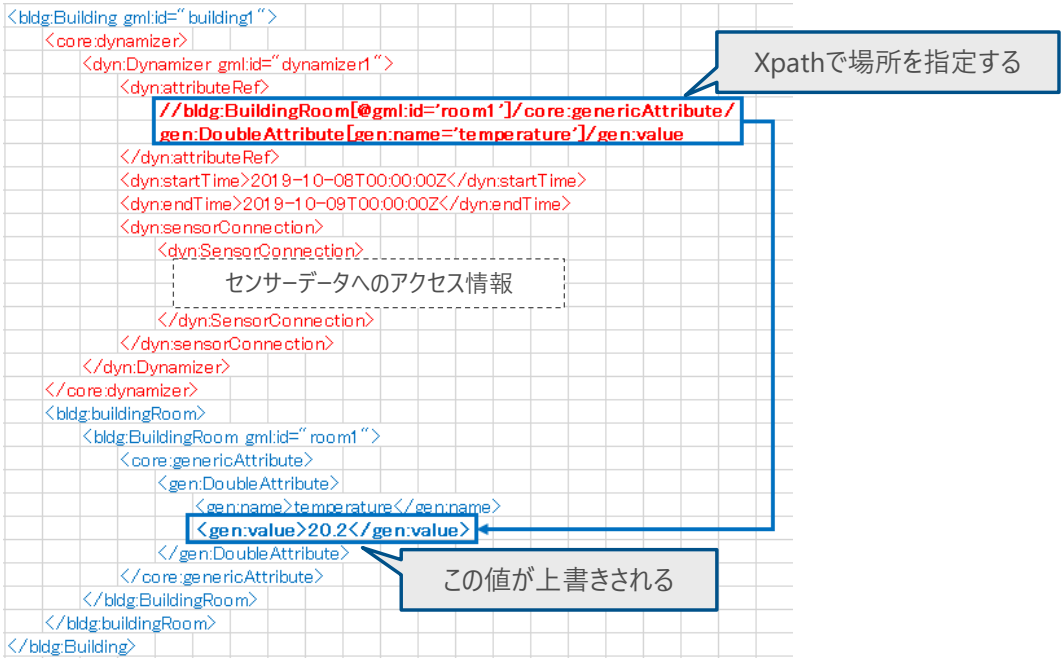



**図 3-9　Dynamizer の出力先の指定**

時間的な属性値の変化は、Dynamizer のデータを読み込むアプリケーションにおいて、入力データとして受け取った時系列データを用いて、出力先となる都市オブジェクトの属性値を上書きすることで表現する。このとき、CityGML ファイルの値を書き換えるのではなく、メモリ上の値を書き換える。

データの実装に必要な交換フォーマットの標準が未発行であるため、CityGML 3.0 に準拠した実装事例は存在しないが、2016 年から 2017 年にかけて実施された OGC の相互運用性プログラムイニシアチブの“Future City Pilot Phase 1 (FCP1)”において、Dynamizer の概念が実装されている。FCP1 では、CityGML 2.0 の ADE（Application Domain Extension）として Dynamizer の UML クラス図による概念モデルと、これに従って CityGML 形式で符号化するための XML Schema が開発された。この検討成果が CityGML 3.0 の Dynamizer モジュールに反映されている。

FCP1 での Dynamizer の実装例を以下に示す。ただし、いずれの事例も現時点ではデモンストレーションシステムにはアクセスできない。

(1)　センサーとセマンティック 3D 都市モデルの統合

イギリスの地図製作機関である Ordnance Survey が実施した、OGC 標準である Sensor Observation Service (SOS)を使用して気象観測所を 3D 都市モデルとリンクさせた例である（図 3-10）。*SensorConnection* を使用してセンサーと接続し、観測結果を取得する。取得した結果をグラフとして視覚化した。

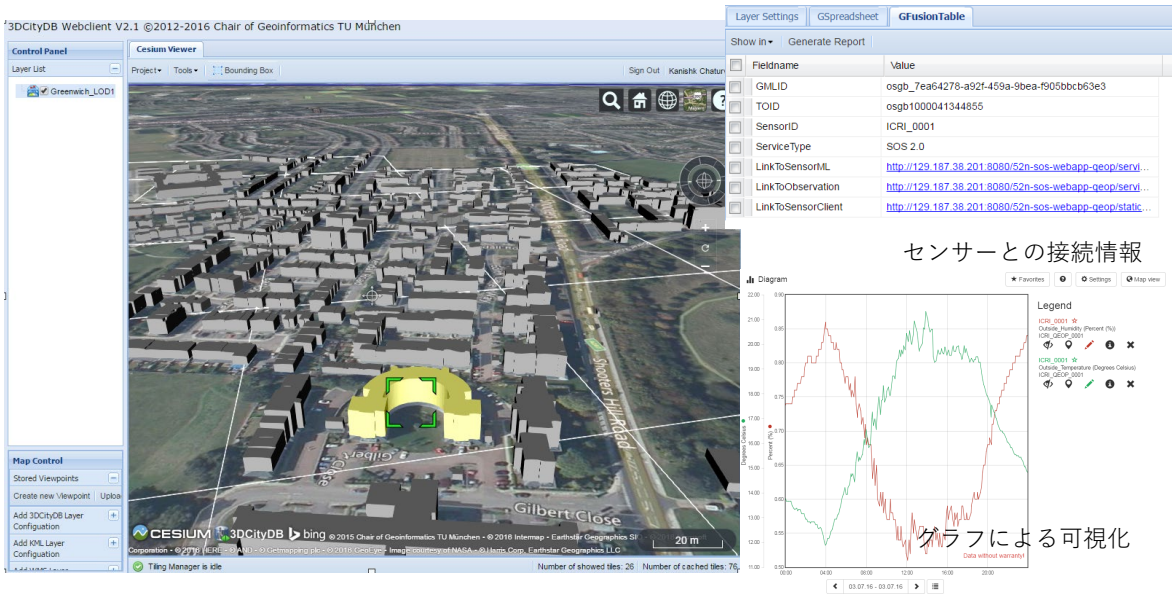
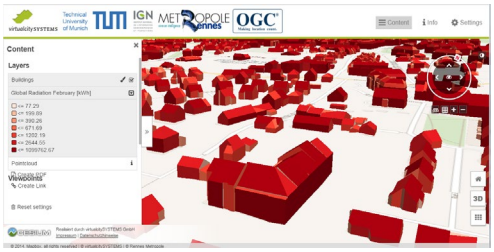


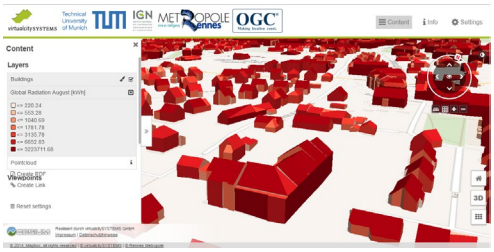

**図 3-10　FCP1 での実装例①（出典[21] ー部補足）**

(2)　センサーとセマンティック 3D 都市モデルの統合

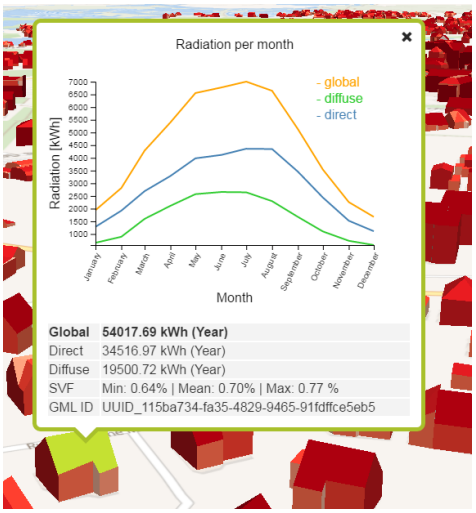
フランス地理院 IGN（Institut Géographique National France）と virtualcitySYSTEMS が実施した、建築物オブジェクトの屋根面・壁面の属性として太陽光発電ポテンシャルの分析結果を付与した例である（図 3-11）。単一の時系列データとして OGC TimeseriesML を使用して屋根面又は壁面の太陽光発電ポテンシャルの分析結果を記録し、その値を用いた色分け表示やグラフ表示を行った。



2 月の日射量による色分け



8 月の日射量による色分け




屋根面の時系列データのグラフ表示

**図 3-11　FCP1 での実装例②（出典[21]　一部補足）**

### 3.4.2.　Dynamizer モジュールの導入に向けた課題

Dynamizer モジュールを使用することにより、都市オブジェクトの時系列変化を記述できるようになる。

ArcGIS や QGIS などの GIS アプリケーションにも時系列データを表現する機能が既に実装されており、時系列データの記述や表現という観点では、CityGML を使用する優位性はない。しかしながら、ArcGIS や QGIS での実装はアプリケーション特有のデータ構造・フォーマットとなるため、アプリケーション間で時系列データをやり取りする際に支障が生じる恐れがある。この場合に、CityGML の Dynamizer モジュールを使用することで、利用環境によらない汎用的なデータとして記述でき、異なるアプリケーション間でのデータ交換が容易となる。

ただし、Dynamizer モジュールが対象とするセンサーの外部ウェブサービスが広範にわたっており、時系列データの記述方法も複数が選択肢として用意されていることから、開発工数や運用の点での負担を考慮すると、OGC TimeseriesML などの国際標準の利用を優先し、対象を絞ってモジュールを利用することが望ましい。

また、CityGML 3.0 が普及するまでには一定の時間がかかると想定される。Dynamizer モジュールの機能を先行して利用したい場合には、FCP1 で開発された Dynamizer モジュールの ADE を使用することで CityGML 2.0 でも Dynamizer モジュールが使用可能となる。FCP1 では、3DCityDB や virtualcitySYSTEMS の実装例があるため、これらの成果も活用することでより効率的に Dynamizer モジュールを実装できる。センサーデータに限らず、例えば、洪水浸水シミュレーションの時系列変化のようなシミュレーションデータも Dynamizer モジュールを使用して記述することができる。洪水浸水シミュレーションの時系列データは、既に一部の都市において 3D 都市モデルとして整備されているが、時間断面ごとの独立したファイルとして作成されている。今後、これらを同一地物の時間変化として Dynamizer モジュールを使って記述・可視化することも考えられる。

## 3.5.　Versioning モジュール

### 3.5.1.　Versioning モジュールの実装方法

Versioning モジュールの実装方法には、更新後の版に全ての地物を含めて記述する方法（`clonePredecessor="false"`）と、変更があった地物のみを記述する方法（`clonePredecessor="true"`）とがある（図 3-12）。

`clonePredecessor` が `false` の場合、バージョンごとにそのバージョンに属する全ての地物を記述するため、各バージョンに属

する地物の把握が容易であり、都市モデルを表示する際にも、ビューワー側は Version に含まれる地物を表示させるだけで良く、ビューワー側の開発負担は少ない。しかしながら、1 ファイルに前バージョンと後続バージョンのデータセットを全て記載すると、データが最低でも 2 時期分となり、データ容量も約 2 倍となる。

一方、clonePredecessor が true の場合、バージョンの間で変更のあった地物のみを記述するため、clonePredecessor が false の場合と比較して、データ容量を押さえることができる。一方、ビューワー側は Transaction から後続バージョンに属する地物を特定しなければならず、バージョンが変わると、Transaction から地物を特定する処理が複雑になる課題が生じる。

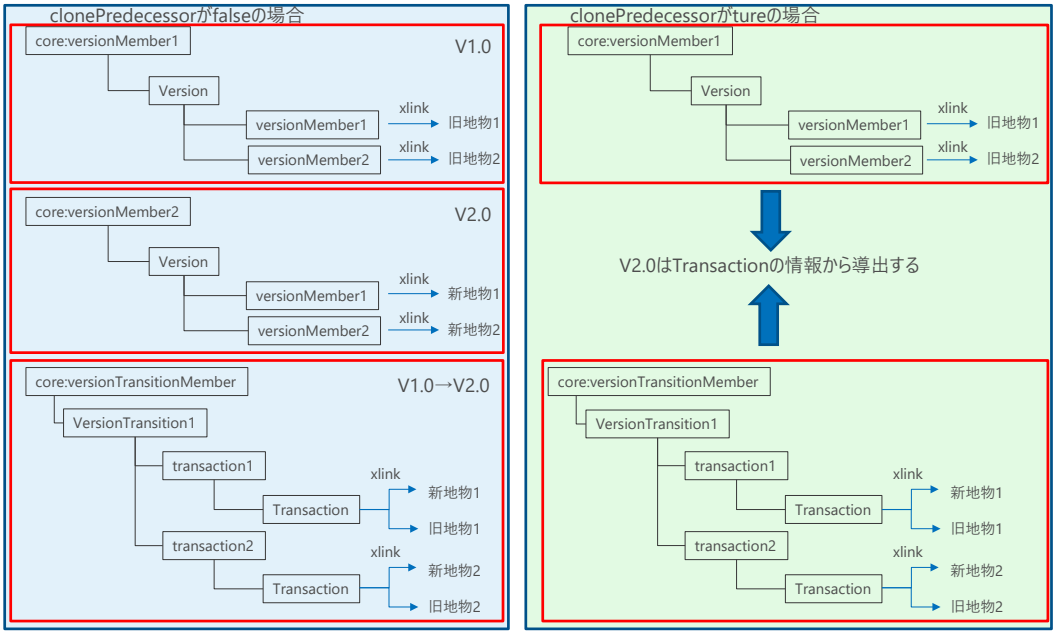



**図 3-12　Versioning モジュールの実装方法**

図 3-13 にサンプルデータ（clonePredecessor="false"）を示す。v1.0 と v2.0 の各バージョン（Version）に、含まれている地物が列記されている（vers:versionMember）。また、更新情報（vers:VersionTransition）として、都市モデル単位のバージョン管理と、地物単位のバージョン管理が記載されている。

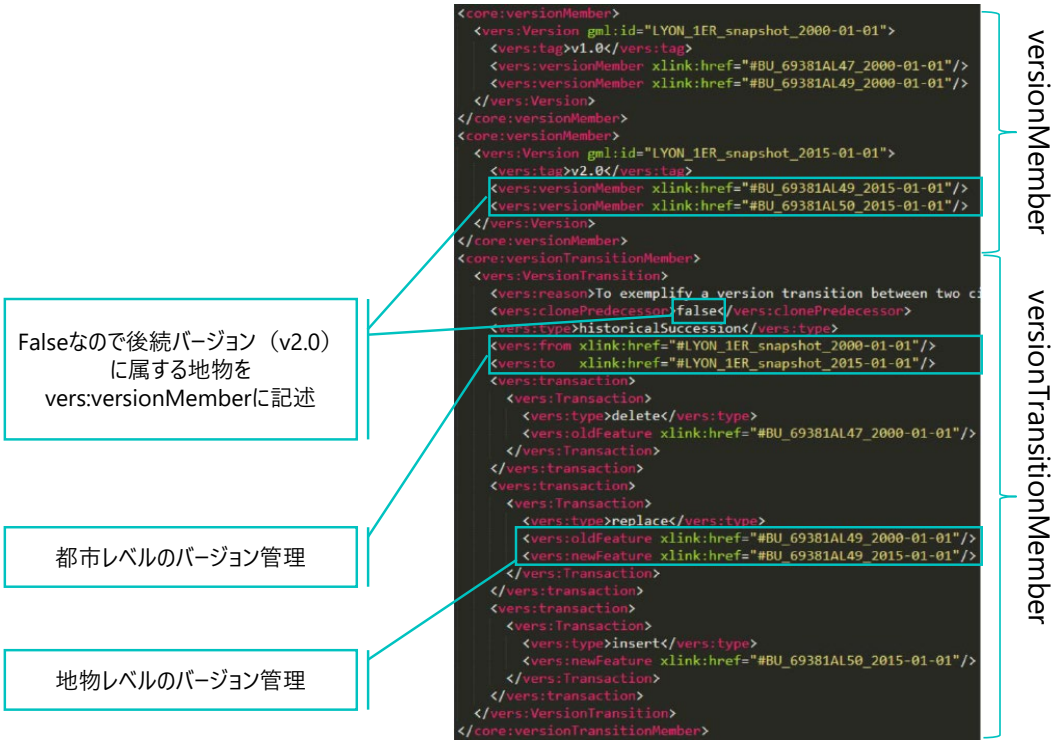



**図 3-13　Versioning のサンプルデータ**

図 3-14 は、バージョン情報に対応する地物のデータである。置換（replace）された古い地物には、データ削除日（terminationDate）が入れられている。


**図 3-14　バージョン管理された地物のサンプルデータ**

サンプルデータでは、Versioning のデータと 3D 都市モデルのデータとを同一のファイルに記述しているが、バージョンが増えることにより、データセットのファイル容量が膨大になる。Versioning のデータと 3D 都市モデルのデータとはファイルを分け、

Versioning のデータから 3D 都市モデルの地物のデータへは外部参照により実装することが現実的である。

なお、Versioning モジュールの実装では、複数の異なるデータセットに含まれる地物の同一性を把握するため、地物の ID 付与が重要である。CityGML SWG の GitHub では ID の付与ルールのベストプラクティスが紹介されている（図 3-15）。

```
<core:cityObjectMember>
    <bldg:Building gml:id="BU_69381AL50_2000-01-01T00-00-00">
        <gml:identifier codeSpace="https://data.grandlyon.com/">BU_69381AL50</gml:identifier>
        <gml:name>Building A</gml:name>
```

**図 3-15　地物の ID の例**

この例では、**gml:identifier** を地物の恒久的な ID とし、異なるバージョンで共通に使用する。この ID は管理される必要があり、**codeSpace** に関連付けなければならない。また、**gml:id** は、データセット固有の ID として使用するが、**gml:identifier** で指定する恒久的な ID に日付や時刻を付与することで効率的にユニークな ID を生成可能である。

### 3.5.2. Versioning モジュールの導入に向けた課題

Versioning モジュールを使用することにより、都市モデルや地物単位のバージョン管理ができるようになる。Versioning モジュールは、更新前後のデータセットや地物を ID により紐づけるデータ構造となっており、データ実装に技術的な支障はない。しかしながら、データ作成費用やデータ量など運用上の支障が生じることが想定される。

これまで、バージョン管理は、都市モデルの場合にはメタデータによりバージョンが把握でき、また、地物単位では更新フラグや更新日のような主題属性を地物に付与することで地物の変化有無を把握することができた。経年変化の把握へのニーズはあるが、地物単位にどう変わったかと分析することは一般的ではなく、空間属性を用いて空間演算により変化した地物の数や変化量を抽出することで代替されることが多かった。変化の有無を把握し前後の地物を紐づける作業は、データ作成費用の増大につながることから、地物ごとの変化を把握するデータが作成されてこなかったともいえる。Versioning モジュールを使用することで、バージョン管理をより詳細に、正確に行うことができるようになるが、データ作成費用に見合った用途やその効果が得られるかを考慮する必要がある。

Versioning モジュールの実装にあたっては、複数バージョンの 3D 都市モデルを格納する場合のフォルダ構成、Versioning モジュールのファイル単位やフォルダ構成、Versioning モジュールの実装範囲を定める必要があり、例えば以下のような運用ルールが考えられる。

- 市区町村ごとのフォルダを作成し、3D 都市モデルの成果品一式をサブフォルダとして格納する。
- udx フォルダに vers フォルダを作成し、主題モジュールごとに作成するバージョン情報を格納する。
- バージョン情報は以下に従い作成する。
  - 3D 都市モデルを整備する場合には **Version** 情報(版の番号と地物の一覧)を作成する。(**clonePredecessor="false"**)
  - 3D 都市モデルを更新する場合には **VersionTransition** により前の版と更新した版を参照し、個々の地物の変遷を **Transaction** で記述する。

図 3-16 に運用ルール例に従いバージョン情報を格納する場合のフォルダ構成を示す。

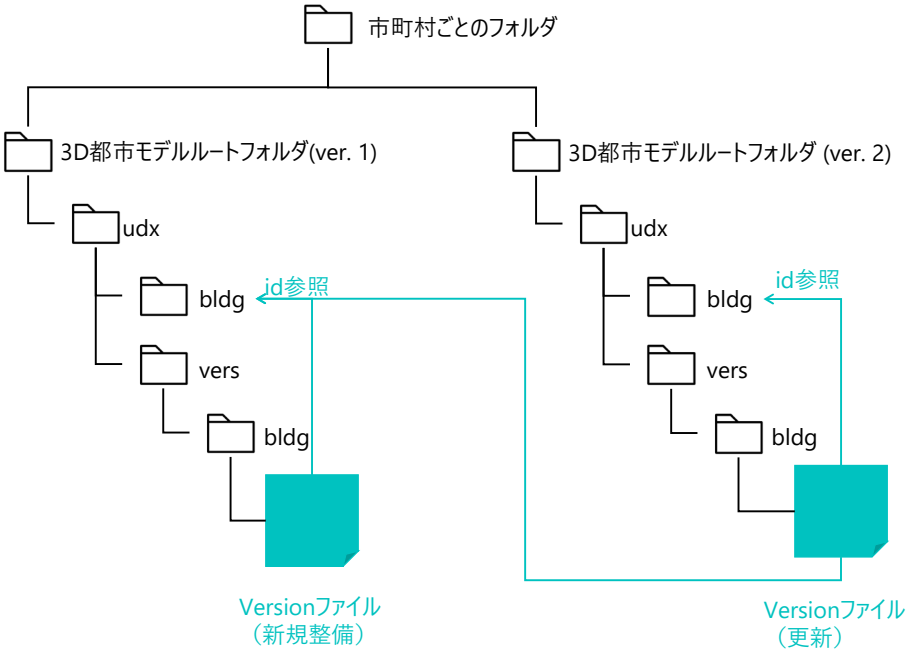



**図 3-16　Versioning モジュールの運用イメージ**

また、Versioning モジュールを使用して明示的に地物の新旧を関連付けるのではなく、新旧の各データセット内で同じ地物が同じ識別子を主題属性としてもつことで、各地物の新旧を把握することができる。例えば、建築物であれば、不動産 ID[9]のような恒久的な識別子を `gml:identifier` に利用することが考えられる。

## 3.6.　PointCloud モジュール

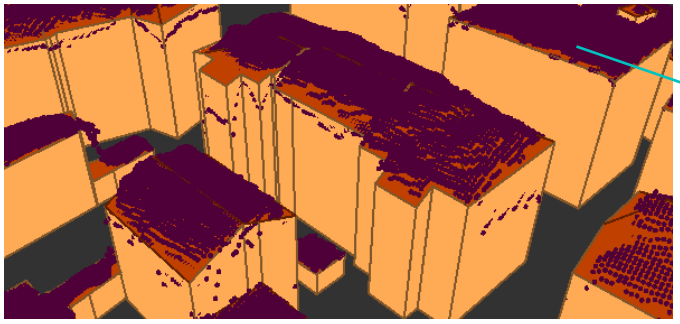
### 3.6.1.　PointCloud モジュール実装方法

PointCloud モジュールを用いて、地物に点群データを関連付ける方法として、CityGML 形式で都市モデルの中に点群データを書き込む方法と、LAS 又は LAZ 形式の点群データのファイルを関連付ける方法がある。

図 3-17 は点群データを、*MultiPoint* を用いて CityGML 形式で記述する方法の実装例である。CityGML 形式で記述されるため、CityGML 対応の可視化ツールで記述された点群データを可視化することができる。一方、個々の点を *Point* として記述し、*MultiPoint* のデータを構成していることから、データ容量が大きくなる。図 3-17 のサンプルデータには建築物のデータ 752 棟が含まれ、各建築物に点群データが記述されており、合計で 448MB となっている。

[9] 国土交通省が推進する、不動産を一意に特定することができる ID であり、「不動産登記簿の不動産番号(13 桁)ー特定コード(4 桁)」で構成される。

```
                    </gml:surfaceMember>
                </gml:MultiSurface>
            </core:lod2MultiSurface>
        </con:WallSurface>
    </core:boundary>
    <core:pointCloud>
        <pcl:PointCloud gml:id="DEBY_LOD2_18271997_fa800d66-0d9e-40c8-82a1-b41ad3d28301">
            <pcl:points>
                <gml:MultiPoint gml:id="DEBY_LOD2_18271997_fa800d66-0d9e-40c8-82a1-b41ad3d28301-0" srsName="urn:adv:crs:DE_DHDN
                    <gml:pointMember>
                        <gml:Point gml:id="DEBY_LOD2_18271997_fa800d66-0d9e-40c8-82a1-b41ad3d28301-1">
                            <gml:pos>4467000.2 5331901.4 533.56</gml:pos>
                        </gml:Point>
                    </gml:pointMember>
```



サンプルデータをFMEで可視化した結果

PointCloudとMultiSurfaceの両方を保持している。

※FZKViewerでも可視化できることを確認

**図 3-17　CityGML 形式で点群データを記述する方法**

図 3-18 は建築物単位に分割された LAZ 形式の点群データファイルを参照する方法である。pcl:pointFile には、点群データファイルへの相対パスが記述されている。点群データを外部ファイルとし、LAZ 形式は点群データの記述に最適化されていることからデータ量は軽減されるが、現状の FZKViewer はこの記述方法に対応しておらず、都市モデルと点群データを同時に表示することができない。

```
                    </gml:MultiSurface>
                </core:lod2MultiSurface>
            </con:WallSurface>
        </core:boundary>
        <core:pointCloud>
            <pcl:PointCloud gml:id="DEBY_LOD2_18271997_24600872-fe6d-48b3-9a34-c37483650e57">
                <pcl:pointFile>..\pointcloud\DEBY_LOD2_18271997.laz</pcl:pointFile>
            </pcl:PointCloud>
        </core:pointCloud>
        <con:height>
            <con:Height>
                <con:highReference>topOfConstruction</con:highReference>
                <con:lowReference>lowestGroundPoint</con:lowReference>
                <con:status>measured</con:status>
                <con:value uom="urn:adv:uom:m">2.877</con:value>
```

**図 3-18　外部ファイルの参照により点群データを記述する方法**

図 3-19 は、外部ファイルである点群データの一部を指定する方法である。pcl:pointFile に URI を記述し点群データを参照している。パラメータを記述することで参照する点群データを指定している。この例の場合では、“point_source_id”が 1 となる全ての点群を参照するように指定されている。

```
                    </gml:surfaceMember>
                </gml:MultiSurface>
            </core:lod2MultiSurface>
        </con:WallSurface>
    </core:boundary>
    <core:pointCloud>
        <pcl:PointCloud gml:id="DEBY_LOD2_18271997_f6b10a4f-242b-4b81-a970-144eb47beb0e">
            <pcl:pointFile>..\pointcloud\4467_5331_40_bDOM_classified_with_point_source_id.laz?idattr=point_source_id&amp;id=1</pcl:pointFile>
        </pcl:PointCloud>
    </core:pointCloud>
```

**図 3-19　外部ファイルの一部を指定し点群データを記述する方法**

点群データファイルには、パラメータで指定された属性“point_source_id”が存在し、同じ建築物に含まれる点には全て同じ値が設定されている。この値を使用し、CityGML ファイルから点群データファイルに含まれる点群を指定している（図 3-20）。

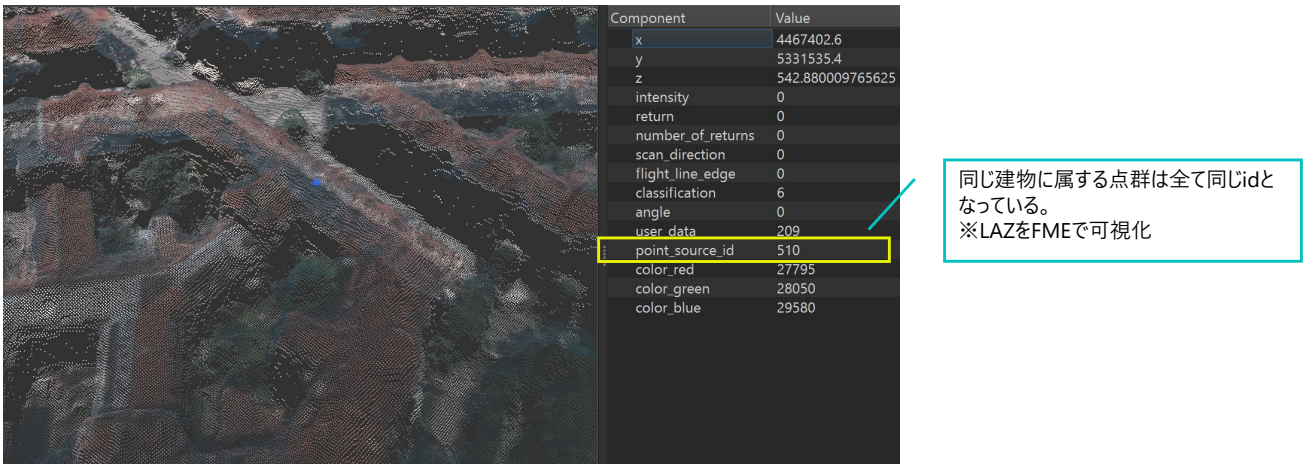



**図 3-20　点群データ（LAZ 形式）の属性**

外部ファイルを参照する方法と同様、現状の FZKViewer はこの記述方法に対応しておらず、都市モデルと点群データとを同時に表示することはできない。

### 3.6.2. PointCloud モジュールの導入に向けた課題

PointCloud モジュールを使用することにより、地物をモデル化する元データとなる点群データを容易に参照することができるようになる。CityGML では地物の幾何形状の表現方法として点、線、面及び立体があることに加え、表現する対象も付属物や開口部といった部品の表現の有無があり、これらの選択肢の組合せにより様々な表現ができる。用途に応じて、適切な組合せを選択できるメリットがあるが、あらかじめ用意されたモデルを使用する場合、そのモデルの表現では不足する場合や過剰な場合が起こりうる。モデル化した地物と点群データをセットにしておくことで、必要な場合に、点群データを用いてモデルを加工したり、新たにモデルを作ったりすることが可能となる。

点群データの表現は、CityGML や外部ファイルなど複数の記述方法が可能となるため、データ整備の観点やソフトウェアの対応などの観点からは、実装方法を限定することが望ましい。データ容量の観点からは点群データを CityGML 形式で記述するよりも、点群に特化したフォーマットを使ったほうが効率的である。国内では、標準的な点群データの形式として、LAS、TXT、CSV 形式が国土地理院の三次元点群データ製品仕様書（案）に明記されている。また、車載写真レーザ測量システムを用いた三次元点群測量マニュアル（案）(国土地理院　令和元年 12 月) においても、データ利用の多様性から LAS 形式が望ましいと記載されている。LAS は OGC 標準となっており、TXT、CSV 形式から LAS 形式に変換することは可能であるため、点群データファイルの外部ファイル形式には LAS 形式を採用することが望ましい。

ただし、LAS 形式にはバージョンがいくつかあり、バージョンによってデータフィールドが異なる。PointCloud モジュールの採用にあたっては、LAS 形式で記述された点群データファイルを適切に読み込むための情報が必要となるが、CityGML 3.0 の PointCloud モジュールには LAS のバージョンや取得した点群データの諸元に関する属性がないため、これらを拡張する必要がある。

## おわりに

本レポートでは、CityGML 3.0 において追加及び改定された内容を整理し、CityGML 3.0 へ移行する利点や移行に向けた課題を明らかにした。以下に主な改定点と移行に向けた留意事項をまとめる。

(1) LOD 定義

各 LOD においてどの地物を使用し、それらをどの空間属性（点、線、面、立体）を使って記述するかの組合せが増えた。各 LOD におけるユースケースや作成手法から取得可能な地物・形状を整理し、現実的な組合せを絞ることが望ましい。特に、CityGML 2.0 には無く CityGML 3.0 で追加されたものは利用環境が整うまでに時間がかかる可能性もあり、採用には注意が必要である。

(2) 地物の追加

IFC、IndoorGML、INSPIRE などの他の標準仕様との整合が考慮され、地物が追加された。同じ事象を、複数の地物で表現できる場合があり、データ作成者によって使用する地物が異なる可能性がある。いずれを使用すべきか、あるいはどのような場合に使い分けるのかという基準を設ける必要がある。

(3) 新規モジュールの採用

新たなモジュールが追加された。特に PointCloud 及び Dynamizer は 3D 都市モデルと動的なデータを連携可能とし、新たなユースケース開発につなげることが期待できる。点群データやセンサーデータにはこれに特化した標準仕様が存在する。データの流通性を考慮すると、点群データやセンサーデータを 3D 都市モデルに統合するよりも、3D 都市モデルを骨格として、各々の標準仕様で記述されたデータと連携できるような構造が効率的である。

CityGML 3.0 では、点群データやセンサーデータとの連携、屋内空間のマルチスケール表現など、3D 都市モデルとしての記述力が高まるとともに、IFC や IndoorGML などの他の標準との互換性も改善された。加えて、概念モデルと符号化仕様が分離されるとともに、概念モデルの冗長性が低減されることで、汎用性も高まった。特に、INSPIRE では、ISO 標準を作業の基礎と位置付けているため、CityGML 3.0 Part 2 において採用される GML 3.2 が ISO 19136 に準拠となることから、CityGML 2.0 を CityGML 3.0 に変換して利用する事例が欧州と中心に増えていくと想定される。

その一方で、データ実装における選択肢が増えたため、選択肢の組合せにより作成される 3D 都市モデルは様々となり、データ作成・利用双方の観点から支障が生じる可能性が高い。選択肢を抽出し、日本における実装仕様を作成することが必要である。実装仕様の作成にあたっては、関連する標準との連携や、CityGML 3.0 対応ツールの開発状況など利用環境を考慮する必要がある。

また、特に日本においてはまだ CityGML が普及しておらず、利用者の知識が不足している。OGC での CityGML 3.0 の開発では、OGC CityGML 3.0 Hackathon や CityGML Challenge を開催し、CityGML 3.0 の仕様を検証するとともに技術者への普及啓発を並行して行っている。同様にして PLATEAU Next（https://www.mlit.go.jp/plateau-next/#schedule-event-diagram）においても、CityGML 形式のデータをハンドリングする機会とコミュニティの醸成が行われている。CityGML の利点は GIS の専門家ではなくても理解可能な形式であることにあり、引き続き CityGML 形式のデータを扱う機会を創出し、技術者や技術者のコミュニティを育成していくことが重要である。

## 謝辞

本技術レポートの作成にあたって、駒澤大学瀬戸寿一准教授、OGC CityGML 仕様策定 WG 副議長石丸伸裕氏、飯嶋孝史氏（Pacific Spatial Solutions 株式会社）に技術的助言をいただきました。心より感謝申し上げます。

## 参考文献


[1] 石丸伸裕・瀬戸寿一（文:西田宗千佳）（2021）デジタルツインのカギを握る国際標準規格「CityGML」の可能性. Plateau [プラトー]国土交通省. https://www.mlit.go.jp/plateau/journal/j003/ [2023.06.19 アクセス]

[2] Kolbe, T.H., Kutzner, T., Smyth, C.S., Nagel, C., Roensdorf,. C. and Heazel, C. (2021) OGC City Geography Markup Language (CityGML) Part 1: Conceptual Model Standard. Version 3.0.0, OGC 20-010. http://www.opengis.net/doc/IS/CityGML-1/3.0 [2023.04.28 アクセス]

[3] Kutzner, T., Smyth, C.S., Nagel, C., Coors, V., Vinasco-Alvarez, D., Ishimaru, N., Yao, Z., Heazel, C., Kolbe, T.H. (2023) OGC City Geography Markup Language (CityGML) Part 2: GML Encoding Standard. Version 3.0, OGC 21-006r2. http://www.opengis.net/doc/IS/CityGML-2/3.0 [予定]
（注：2023 年 6 月 30 日時点では、https://docs.ogc.org/is/21-006r2/21-006r2.html よりアクセス可）

[4] OGC, opengeospatial/CityGML3.0-GML-Encoding. https://github.com/opengeospatial/CityGML3.0-GML-Encoding [2023.06.19 アクセス]

[5] Gröger, G., Kolbe, T.H., Nagel, C. and Häfele, K.H. (2012) OGC City Geography Markup Language (CityGML) Encoding Standard. Version 2.0.0, OGC 12-019. http://www.opengis.net/spec/citygml/2.0 [2023.04.28 アクセス]

[6] Kutzner, T., Chaturvedi, K., Kolbe, T.H.(2020) CityGML 3.0: New Functions Open Up New Applications. PFG 88, 43–61. https://doi.org/10.1007/s41064-020-00095-z [2023.04.28 アクセス]

[7] 石丸伸裕（2014）3 次元地理空間データ CityGML/IndoorGML に関する国際標準化活動. 「地図」, 52 (3), 29-36. https://www.jstage.jst.go.jp/article/jjca/52/3/52_3_29/_pdf [2023.06.19 アクセス]

[8] 瀬戸寿一（2020）3D 都市モデルの CityGML の現在とこれから. Qiita. https://qiita.com/tosseto/items/34a2a801b8eb299453d1 [2023.06.19 アクセス]

[9] Nega, A., Coors, V (2022) THE USE OF CITYGML 3.0 IN 3D CADASTRE SYSTEM: THE CASE OF ADDIS ABABA CITY. Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci., XLVIII-4/W4-2022, 109–116, https://doi.org/10.5194/isprs-archives-XLVIII-4-W4-2022-109-2022 [2023.06.19 アクセス]

[10] 国土交通省都市局 (2023) 3D 都市モデル整備のための BIM 活用マニュアル. https://www.mlit.go.jp/plateau/libraries/handbooks/ [2023.06.19 アクセス]

[11] Sun, Qun, Xiaoguang Zhou, and Dongyang Hou (2020) A Simplified CityGML-Based 3D Indoor Space Model for Indoor Applications. Applied Sciences 10, no. 20: 7218. https://doi.org/10.3390/app10207218 [2023.06.19 アクセス]

[12] Beil, C., Ruhdorfer, R., Coduro, T., Kolbe, T.H. (2020) Detailed Streetspace Modelling for Multiple Applications: Discussions on the Proposed CityGML 3.0 Transportation Model. ISPRS International Journal of Geo-Information 9, no. 10: 603. https://doi.org/10.3390/ijgi9100603

[13] 渡辺信之, 大木章一, 中村孝之 (2004). 海外地図作成機関における写真測量技術に関する動向調査. 国土地理院時報（2004, 105 集）. https://www.gsi.go.jp/REPORT/JIHO/vol105-content105.html. [2023.06.19 アクセス]

[14] 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, https://doi.org/10.5194/isprs-annals-IV-4-W5-9-2017, 2017. [2023.04.28 アクセス]

[15] INSPIRE (2013) INSPIRE Data Specification on Buildings – Technical Guidelines, https://inspire.ec.europa.eu/id/document/tg/bu [2023.04.28 アクセス]

[16] Heazel,C.(2021) OGC City Geography Markup Language (CityGML) 3.0 Conceptual Model Users Guide. Version 1.0, OGC 20-066, http://www.opengis.net/doc/UG/CityGML-user-guide/3.0 [2023.04.28 アクセス]

[17] Ledoux, H. (2021) CityJSON Community Standard 1.0. OGC 20-072r2. http://www.opengis.net/doc/CS/CityJSON/1.0 [2023.06.19 アクセス]

[18] Ledoux,H., Ohori, K.A., Kumar, K., Dukai, B., Labetski., A., Vitalis,S. (2019) CityJSON: a compact and easy-to-use encoding of the CityGML data model. Open geospatial data, softw. stand. 4, 4. https://doi.org/10.1186/s40965-019-0064-0 [2023.06.19 アクセス]

[19] Yao, Z., Nagel, C., Kunde, F., Hudra, G., Willkomm, P., Donaubauer, A., Adolphi, T., Kolbe, T.H (2018) 3DCityDB - a 3D geodatabase solution for the management, analysis, and visualization of semantic 3D city models based on CityGML. Open Geospatial Data, Software and Standards volume 3, Article number: 5. https://doi.org/10.1186/s40965-018-0046-7 [2023.06.19 アクセス]

[20] Chaturvedi, K., Kolbe, T.H. (2016), Integrating dynamic data and sensors with semantic 3D city models in the context of smart cities. In: Proceedings of the 11th international 3D geoinfo conference, Athens, ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol IV-2/W1. https://doi.org/10.5194/isprs-annals-IV-2-W1-31-2016 [2023.04.28 アクセス]

[21] Chaturvedi, K., Kolbe, T.H. (2017) Future City Pilot 1 Engineering Report, OGC 16-098 http://docs.opengeospatial.org/per/16-098.html [2023.04.28 アクセス]

[22] Chaturvedi, K. (2021) Integration and Management of Time-dependent Properties with Semantic 3D City Models, https://www.researchgate.net/publication/353794195_Integration_and_Management_of_Time-dependent_Properties_with_Semantic_3D_City_Models

[23] OGC. CityGML 3.0 Versioning Module: Basic Versioning Example, https://github.com/opengeospatial/CityGML-3.0Encodings/blob/master/Moved_to_CITYGML-3.0Encoding_CityGML/Examples/Versioning/Basic%20examples/CityGML%203.0%20-%20Versioning%20Module.pdf [2023.04.28 アクセス]