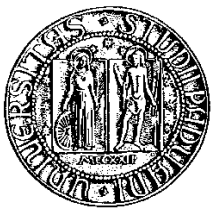

Università degli Studi di Padova

Facoltà di Ingegneria

Corso di Laurea Specialistica in Ingegneria Informatica

tesi di laurea

Progettazione e sviluppo di un toolkit per la gestione di dati spaziali 3D nei formati standard OGC CityGML e KML per il geodatabase opensource PostGIS

Relatore: prof. Massimo Rumor

Laureando: Paolo Varagnolo

18 aprile 2011

Sommario

I modelli 3D per la rappresentazione virtuale di città e aree urbane stanno trovando in questo ultimo decennio un forte sviluppo e rappresentano un importante sorgente di dati per un sempre maggiore numero di applicazioni. Tra questi particolare importanza assumono i modelli standard OGC KML e CityGML.

KML sviluppa principalmente la visualizzazione dell'informazione, codifica e trasporta dati geografici 3D e 2D di oggetti per visualizzarli in un browser, è il formato utilizzato da Google Earth.

CityGML prevede classi e relazioni per i più rilevanti oggetti topografici urbani e consente la descrizione delle proprietà geometriche, topologiche, semantiche e visuali. Le informazioni tematiche messe a disposizione fanno si che CityGML non sia semplicemente un formato per lo scambio e la visualizzazione di dati 3D dei modelli urbani, ma permetta di usare tali modelli per analisi più sofisticate in ambiti applicativi diversi.

É importante gestire, manipolare la mole di dati prodotta da questi modelli in modo adeguato. Per questo si rende necessario utilizzare DBMS dotati di estensione spaziale. Il progetto consiste nel disegno di un modello dati per la gestione di modelli CityGML, e la realizzazione di un toolkit che permetta di importare e esportare istanze di modelli CityGML. Inoltre si è sviluppato un modulo per l'estrazione di modelli KML attraverso una opportuna mappatura dei dati gestiti nel GeoDatabase in formato CityGML.

Tutti i componenti della piattaforma software utilizzata sono FOSS.

Indice

Sommario

Capitolo 1

Introduzione

I modelli 3D per la rappresentazione virtuale di città e aree urbane stanno trovando in questo ultimo decennio un forte sviluppo e rappresentano un'importante sorgente di dati per un sempre maggiore numero di applicazioni. Sia a livello pubblico che privato, alcune organizzazioni stanno già pensano ad introdurre la terza dimensione e la modellazione solida degli oggetti nei propri GeoDatabase-2D.

Le applicazioni che possono beneficiare di modelli 3D della struttura urbana sono, tra le altre, applicazioni per la pianificazione urbana, per la gestione delle utilities, per la gestione del rischio e delle emergenze, la sicurezza nelle abitazioni e negli ambienti di lavoro.

Tutte queste applicazioni richiedono, oltre alla componente geometrica, molte informazioni sugli oggetti, impongono un livello di dettaglio elevato nonché la strutturazione degli oggetti in parti componenti.

Un modello dei dati che soddisfa tutte queste premesse è lo standard OGC CityGML. CityGML è un modello specificatamente progettato per la rappresentazione di oggetti urbani in 3D, definisce le classi e le relazioni e prevede proprietà geometriche, topologiche, semantiche e visuali.

É importante gestire, manipolare la mole di dati prodotta da questi modelli in modo adeguato. Per questo si rende necessario utilizzare DBMS dotati di estensione spaziale. Il progetto consiste nel disegno di un modello dati per la gestione di modelli CityGML, e la realizzazione di un toolkit che permetta di importare e esportare istanze di modelli CityGML. Inoltre si è sviluppato un modulo per l'estrazione di modelli KML attraverso una opportuna mappatura dei dati gestiti nel GeoDatabase in formato CityGML.

Il progetto ha come obbiettivo quello di utilizzare solamente software opensource. Per sviluppare il toolkit si è scelto l'ambiente JAVA, mentre come DBMS spaziale si è scelto PostgreSQL con estensione spaziale PostGIS che permette la gestione di dati spaziali, nonché la gestione dei modelli in modo flessibile, veloce ed efficiente.

Capitolo 2

Obiettivi

L'obbiettivo di questo lavoro è progettare e sviluppare un ambiente per l'importazione e l'esportazione in un GeoDatabase opensource di istanze di modelli in formato CityGML. In fase di esportazione si è deciso di considerare, non solo il formato CityGML ma anche il molto usato formato KML (supportato da Google).

La prima parte della progettazione è stata quella di analizzare il progetto pilota 3D City DB sviluppato dal IGGS della Technische Universität Berlin che prevede la gestione di modelli CityGML con il DBMS Oracle Spatial 10G R2 ed ha sviluppato un toolkit per questo DBMS.

Si è deciso di effettuare il porting completo del progetto 3D City DB in modo da poter utilizzare un DBMS opensource. La scelta del DBMS è caduta su Postgresql/PostGIS. PostgreSQL/PostGIS presenta caratteristiche adeguate di flessibilità ed affidabilità, Grazie all'estensione spaziale PostGIS fornisce inoltre i tipi di dati specificati negli standard dell'Open Geospatial Consortium. Si tratta quindi di definire lo schema in modo che sia conforme allo standard CityGML e per questo è necessario ridisegnare lo schema del database realizzato nel progetto 3D City DB per adattarlo a PostgreSQL/PostGIS.

Per gestire l'importazione e l'esportazione dei dati è necessario fare il porting del toolkit per ottenere un frontend tra l'utente e il database spaziale scelto. In dettaglio il toolkit permette di importare ed esportare le istanze di modelli CityGML, deve anche estrarre i modelli in formato KML trasformandoli dal formato nativo CityGML. Sia in fase di importazione che in fase di esportazione deve essere possibile filtrare i dati in base ad alcuni criteri:

- Importare/Esportare solo elementi contenuti parzialmente o interamente in una certa regione delimitata da un opportuno bounding box.

- Importare/Esportare solo elementi appartenenti a specifiche categorie di feature indicate.

- Importare/Esportare solo elementi con un certo gml_id o gml_name.

Inoltre in fase di esportazione deve essere possibile effettuare una suddivisione del modello in parti per poter selezionare parti di interesse e per ridurre la dimensione.

Il toolkit deve essere un software realizzato con una architettura multithread necessaria per avare alte prestazioni e poter importare ed esportare file di grandi dimensioni.

Capitolo 3

Protocolli Standard OGC utilizzati per il GIS 3D

L'Open Geospatial Consortium è un consorzio internazionale no-profit costituito da aziende private, agenzie governative ed università che si prefigge di sviluppare delle regole standard per i servizi geospaziali. Questi sono classificabili come sistemi software che scambiano dati sul protocollo HTTP. L'interazione tra i servizi e le applicazioni avviene mediante l'invio di messaggi XML. Il sistema di comunicazione garantisce l'interoperabilità essendo indipendente dalla piattaforma hardware, dal sistema operativo e dal formato originario dei dati: qualsiasi software client e server possono comunicare tra di loro purché implementino in modo corretto gli standard. L'interscambio di dati geo-spaziali avviene tra diversi elaboratori facenti parte della stessa rete o facenti parte di reti diverse comunicanti tra loro (World Wide Web).

3.1 OGC CityGML

CityGML nasce come metodo generale per la strutturazione di modelli di ambienti urbani con scopi di visualizzazione comprendendo oltre alle geometrie degli oggetti anche le loro proprietà semantiche e tematiche.

Definisce classi e relazioni per gli oggetti topografici più rilevanti nei modelli di città e regioni riguardanti le loro proprietà geometriche, topologiche, semantiche e il loro aspetto esterno.

Sono incluse generalizzazioni gerarchiche tra classi tematiche, aggregazioni, relazioni tra oggetti e proprietà spaziali. Le informazioni tematiche messe a disposizione fanno si che CityGML non sia semplicemente un formato per lo scambio e la visualizzazione di dati 3D dei modelli urbani, ma permette di usare tali modelli per analisi più sofisticate in ambiti applicativi diversi: simulazioni, data mining urbano, informazioni tematiche.

CityGML è un modello di dati aperto e basato su XML per memorizzare e scambiare dati spaziali 3D. Si tratta di uno schema basato sulla versione 3.1.1 di Geography

Markup Language (GML3), standard internazionale per lo scambio di dati territoriali rilasciato dall'Open Geospatial Consortium (OCG) e la ISO TC211.

L'obiettivo di CityGML è quello di raggiungere una definizione comune delle entità di base, attributi e relazioni di un modello 3D di una città. Questo è particolarmente importante per quanto riguarda la manutenzione e sostenibilità dei modelli 3D, consentendo il riutilizzo degli stessi dati in diversi campi applicativi.

Le parti principali di questo linguaggio sono il modello geometrico e il modello tematico. Il primo rappresenta tutte le informazioni di tipo geometrico e topologico in tre dimensioni degli oggetti del modello, mentre il secondo include informazioni di tipo semantico, che aiutano l'utente a stabilire le relazioni tra i vari oggetti del modello, e utilizza il modello geometrico per modellare diversi oggetti tematici, ad esempio: "*Digital Terrrain model*", vegetazione [*vegetation*] (oggetti isolati e a allo stesso tempo una loro aggregazione), bacini idrici [*water bodies*], rete stradale[*transportation facilities*] e arredo urbano[*city furniture*].

Tutto ciò che non è esplicitamente modellato dallo standard è modellabile attraverso il concetto di oggetti generici e da attributi. Oggetti che hanno stessa forma e che appaiono molte volte anche in posizioni diverse, come gli alberi, possono essere modellati come prototipi e usati più volte nel modello. Inoltre il concetto di *TerrainIntersectionCurve* è introdotto per integrare oggetti 3D con il *Digital Terrain Model* per avere le posizioni corrette al fine di evitare ad esempio edifici galleggianti e affossamenti nel terreno.

CityGML distingue cinque diversi livelli di dettaglio (LoD), in cui gli oggetti, all'aumentare del livello di dettaglio, diventano sempre più dettagliati sia dal punto di vista geometrico che tematico. Inoltre, gli oggetti possono avere delle referenze esterne corrispondenti ad oggetti in dataset esterni. Infine, agli oggetti CityGML è possibile assegnare un aspetto esteriore (appearances). Le apparenze non sono solo proprietà visive ma rappresentano anche proprietà osservabili arbitrarie delle superfici quali rumore, l'inquinamento o problemi strutturali indotti, ad esempio, da un terremoto.

3.1.1 Caratteristiche Generali

Modularizzazione Il modello CityGML consiste nella definizione di classi per i più importanti tipi di oggetti del modello cittadino 3D. Queste classi sono state individuate per essere necessarie ed importanti in molte applicazioni.

Il modello CityGML è scomposto in *core module* e *thematic extension modules*. Il *core module* comprende i concetti e le componenti di base di CityGML e, quindi, deve essere implementato da qualsiasi sistema. Basati sul *core module*, ogni estensione copre una tematica specifica di un campo del modello virtuale 3D della città. CityGML introduce i seguenti dieci moduli tematici di estensione: *Appearance*, *Building*, *CityFurniture*, *CityObjectGroup*, *Generics*, *LandUse*, *Relief*, *Transportation*, *Vegetation*, *WaterBody*.

Le implementazioni di CityGML possono supportare qualsiasi combinazione dei moduli di estensione in combinazione con il *core module*. Tali combinazioni di mo-

duli sono denominati *CityGML profiles*. Pertanto, CityGML consente di implementare anche solo parzialmente il modello globale CityGML in modo valido.

Modellazione Multi-Scale (5 livelli di dettaglio, LoD) CityGML supporta cinque diversi livelli di dettaglio (LoD). All'aumentare del livello di dettaglio gli oggetti diventano sempre più particolareggiati sia per quel che riguarda la geometria sia per la tematica.

I diversi LoD sono:

LoD 0 modello regionale (2.5D modello di terreno);

LoD 1 città/modello del sito (modello di blocco con o senza tetti);

LoD 2 città/modello del sito (texture di tetti e delle facciate);

LoD 3 città/modello del sito (modello architetturale dettagliato);

LoD 4 modello dell'interno (navigazione all'intero dell'edificio).

Ogni file CityGML può, ma non deve necessariamente, contenere rappresentazioni multiple per ogni oggetto in un diverso livello di dettaglio simultaneamente.

Figura 3.1: Esempio di LOD

Terrain Intersection Curve (TIC) Una questione cruciale nella modellazione della città è l'integrazione di oggetti 3D e il terreno. I problemi sorgono se gli oggetti 3D galleggiano sopra o sprofondano nel terreno. Ciò è particolarmente vero nel caso in

cui si combinino terreni e oggetti 3D in diversi livelli di dettaglio. Per superare questo problema è stato introdotto il *TerrainIntersectionCurve* (TIC) di un oggetto 3D. Queste curve indicano la posizione esatta, dove il il terreno e l'oggetto 3D si incontrano(vedi Figura **??**. Queste informazioni possono essere utilizzate per integrare l'edificio e un terreno, "tirare su" o "tirare giù" il terreno circostante, in modo da adattarsi alla *TerrainIntersectionCurve*.

Il DTM può essere localmente deformato per adattarsi alle TIC. In questo modo, il TIC assicura anche la corretta posizione delle texture o l'abbinamento delle texture dell'oggetto con il DTM. Dal momento che l'intersezione con il terreno può variare a seconda del livello di LOD, un oggetto 3D può avere *TerrainIntersectionCurves* diversi per tutti i LOD.

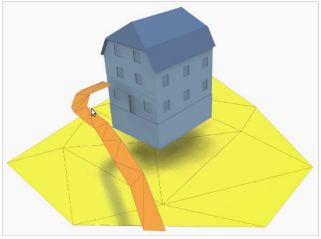

Figura 3.2: Esempio senza TIC.

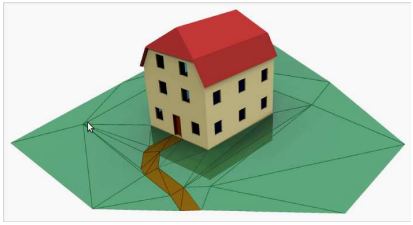

Figura 3.3: Esempio con TIC.

Riferimenti Esterni Gli oggetti 3D sono spesso derivati da oggetti presenti in altri database, (ad esempio quelli del catasto e in un database con modelli architettonici). L'aggiornamento di un riferimento dovrà essere propagato al fine di fornire tali informazioni. Ogni *CityObject* può avere riferimenti esterni indicati come un *Uniform Resource Identifier* (URI), un formato generico per i riferimenti ad ogni tipo di risorse su Internet. Il concetto generico di riferimenti esterni permette a qualsiasi CityObject un numero arbitrario di collegamenti a oggetti corrispondenti a sistemi informatici esterni.

City object groups Il concetto di raggruppamento in CityGML permettere l'aggregazione di oggetti secondo criteri definiti dall'utente, e per rappresentare e trasferire queste aggregazioni come parte di un modello unico. Un gruppo può contenere altri gruppi come membri, permettendo raggruppamenti nidificati con profondità arbitraria. Il concetto raggruppamento è espresso dall'estensione *CityObjectGroup* di CityGML.

Prototypic objects / scene graph concepts In CityGML oggetti con forma uguale come gli alberi, semafori, segnali stradali, ecc. possono essere rappresentati come **prototipi** che vengono instanziati più volte in luoghi diversi. La geometria del prototipi è definita in un sistema di coordinate locali. Ogni istanza è rappresentata da un riferimento al prototipo, un punto base del sistema di coordinate di riferimento e una matrice di trasformazione che facilita il ridimensionamento, rotazione, e la traduzione del prototipo. Il principio è adottato dal concetto di scena utilizzato negli standards

della computer graphics come VRML e X3D. Poiché GML3 non fornisce il supporto per la scena, questo è implementato come estensione di GML3.

3.1.2 Coerenza nella modellazione geometrica e semantica

Uno dei più importanti principi di progettazione di CityGML è la coerenza tra la modellazione semantica e le proprietà geometriche/topologiche.

A livello semantico gli oggetti del mondo reale sono rappresentati da 'cose' come: edifici, muri, finestre o stanze. La loro descrizione include: attributi, relazioni ed aggregazioni gerarchiche tra oggetti. Quindi la parte di relazioni tra gli oggetti può essere derivata solo a livello semantico e non geometrico. A livello spaziale, gli oggetti geometrici sono assegnati ad oggetti che rappresentano la loro locazione ed estensione nello spazio.

Quindi il modello è composto da due livelli gerarchici:

- semantico
- geometrico

in cui gli oggetti corrispondenti sono legati tra loro da relazioni di tipo gerarchico.

Il vantaggio di questo approccio consiste nel fatto che è possibile effettuare una navigazione in entrambe le gerarchie e tra le gerarchie arbitrariamente, per poter eseguire query tematiche e/o geometriche. Infine, se per un determinato oggetto esistono entrambe le gerarchie deve essere garantita la loro coerenza. Ad esempio se un muro, a livello semantico, ha una porta e una finestra, a livello geometrico la geometria che rappresenta il muro dovrà contenere anche le geometrie che rappresentano la finestra e la porta.
Il modello semantico di CityGML consiste nella definizioni di classi per le funzioni più importanti all'interno di modelli cittadini, compresi edifici, DTM, bacini idrici, trasporti, vegetazione, e infrastrutture. La Figura3.4 mostra una piccola parte del modello semantico utilizzato per descrivere gli edifici. Tutte le classi sono derivate dalla classe base *Feature*, definita nella norma ISO 19.109 ed in GML3 per la rappresentazione degli oggetti territoriali e delle loro aggregazioni.
Le seguenti osservazioni possono essere fatte dalla Figura 3.4:

- Un edificio può essere composto in modo ricorsivo da altre costruzione;
- Un edificio può essere delimitato da diversi tipi di superfici (Pareti, tetto), che possono avere aperture come finestre e porte;
- Un edificio può avere altri edifici esterni;
- Entrambi i modelli permettono aggregazioni in più livelli.

Le proprietà spaziali delle *Feature* di CityGML sono rappresentate dal modello di geometria di GML3, basato sullo standard ISO 19.107 "Spatial Schema" (Herring, 2001), rappresenta la geometria 3D in base al well-known Boundary Representation (B-Rep,

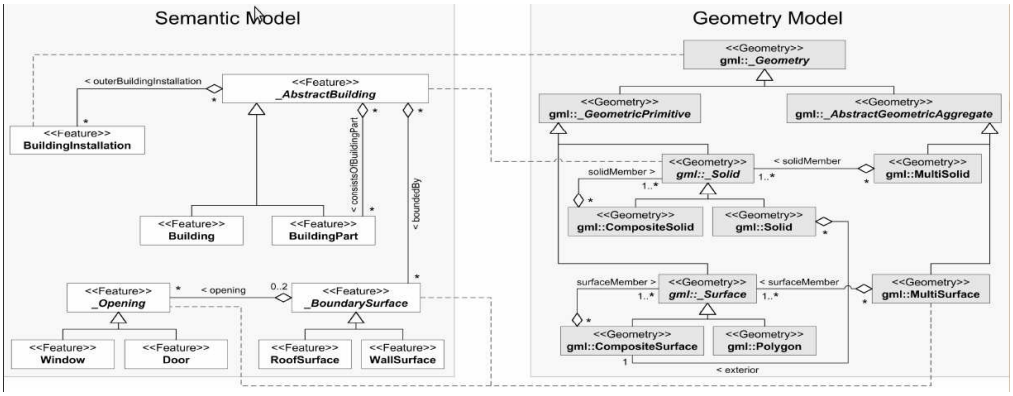

Figura 3.4: UML class diagram of CityGML's semantic and geometry model

Foley et al., 1995). CityGML effettivamente utilizza solo un sottoinsieme delle geometrie di GML3.
In CityGML, la topologia può essere rappresentata in modo esplicito. Ogni parte dello spazio può essere modellato solo una volta e poi si fa riferimento con tutte le funzionalità che comprendono la stessa geometria.

Coerenza tra modello spaziale e semantico Esiste una duplice struttura che comprende sia la geometria che la semantica, questa viene realizzata da due aggregazioni gerarchie di funzionalità e geometria. Collegandole ai corrispondenti oggetti (rappresentati da linee tratteggiate nelle Figura 3.4 e 3.5 a 3.9), la coerenza nella modellizzazione tra la semantica e la geometria è assicurata.

La coerenza nel contesto geospaziale descrive la consistenza delle relazioni spaziali con le entità semantiche. Le relazioni sono realizzate in forma di associazioni, che possono essere stabilite solo in caso di somiglianza strutturale. Solo allora la semantica e le informazioni spaziali possono essere utilizzate insieme, avendo due evidenti benefici :

- Gli oggetti geometrici conoscono chi sono.
- Le entità semantiche conoscono dove sono e quali sono le loro aree.

In un certo senso, la somiglianza strutturale tra scomposizioni spaziali e semantica viene descritta da un omomorfismo tra le due strutture. Fondamentalmente, maggiori sono le relazioni di aggregazioni del modello reale che posso essere mappate e più alto è il grado di coerenza.

In seguito si esaminerà, distinguendo diversi casi di modelli 3D rispetto al loro grado di complessità semantica e spaziale. La complessità spaziale non si riferisce al numero di oggetti geometrici primitivi, piuttosto denota la suddivisione della geometria strutturale in parti significative, definendo le varie gerarchie. Analogamente si può dire per la complessità semantica, rispetto alla suddivisione delle informazioni.

Da Figura 3.5 a Figura 3.9 rappresentano un edificio composto da muri, finestre, porte, tetto, e scale. Le immagini sul lato sinistro mostrano l'aspetto visivo (uguale per tutti gli esempi) al centro e sul lato destro viene rappresentata la struttura semantica e spaziale per descrivere l'edificio. Le linee tratteggiate segnano le relazioni.

Caso 1: Solo geometria, no semantica. Il primo caso (vedi Figura 3.5), descrive i tipici modelli 3D come VRML, X3D, KML, U3D. Questi modelli comprendono una geometria più o meno strutturata. Questi sono output comuni per la modellazione 3D utilizzati in computer grafica e CAD. Poiché essi non comprendono informazioni semantiche non c'è alcuna coerenza.

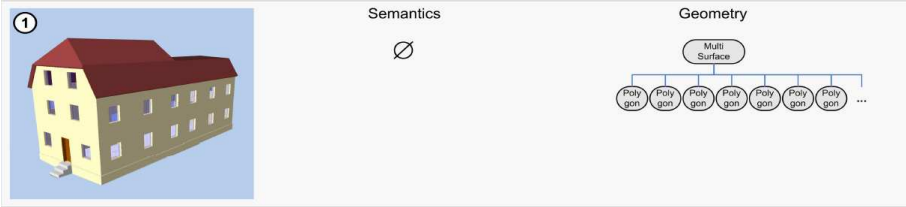

Figura 3.5: Caso 1

Caso 2: Solo semantica, ma non geometria. Questo caso è piuttosto insolito e descrive una situazione in cui è noto che i modelli cittadini sono costituiti da specifiche caratteristiche geospaziali conosciute, dove la geometria è sconosciuta o non disponibile.

Caso 3: oggetti semplici, con geometria non strutturata(Figura 3.6). Gli oggetti sono rappresentati da elementi geografici. Ogni elemento ha un attributo spaziale consistente in un raccolta non strutturata di superfici 3D e possibilmente un numero scalare di attributi non-spaziali. Questo modello ha un alto grado di coerenza, se la morfologia descritta dalla geometria è semplice, è poco coerente se la geometria non strutturata è descritta con una forma complessa (ad esempio un edificio con dettagli complessi).

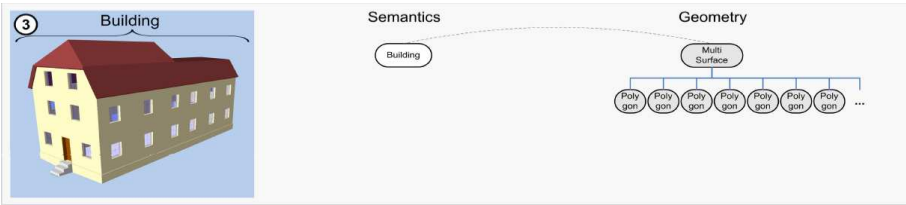

Figura 3.6: Caso 3

Caso 4: oggetti semplici, con geometria strutturata (Figura 3.7). In questo caso, la geometria non solo è dettagliata, ma anche strutturata rispetto alla decomposizione spaziale. Tuttavia, riguardo la semantica è indicata solo l'esistenza di un edificio. Quindi, le relazioni non possono essere stabilite tra le sub-geometrie e le componenti semantiche mancanti con conseguente basso grado di coerenza.

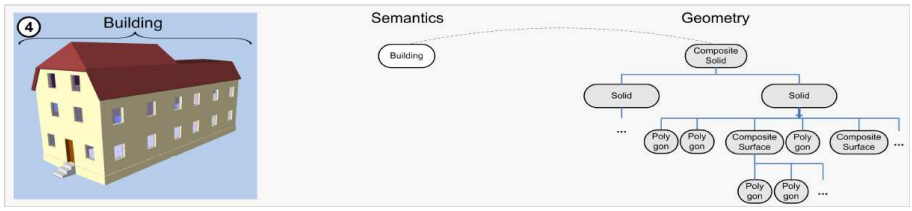

Figura 3.7: Caso 4

Caso 5: oggetti complessi con geometria non strutturata (Figura 3.8).La semantica è dettagliata, questo significa che la scomposizione tematica dell'edificio è nota. Dato che la geometria è dettagliata, ma non è strutturata, le relazioni tra semantica e geometria non possono essere stabilite nei vari livelli di aggregazione. Così, la coerenza è data solo dall'edificio in quanto oggetto complessivo, rappresentato dall'insieme di superfici 3D.

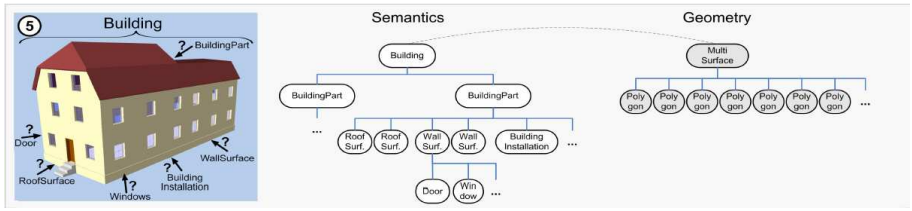

Figura 3.8: Caso 5

Caso 6: oggetti complessi, con geometria strutturata (Figura 3.9). Entrambi i modelli semantico e geometrico sono descritti da complesse aggregazioni. Se tutti i componenti semantici vengono correlati ai rispettivi componenti geometrici sullo stesso livello di gerarchia, la struttura è considerata pienamente coerente (Figura 3.9). Questi modelli mostrano il più alto grado di qualità strutturale poiché entrambi ricchi dal punto di vista semantico e geometrico, e sono strutturalmente isomorfi.

Dalle descrizioni appena effettuate risulta chiaro che oltre agli aspetti spaziali e semantici anche la coerenza delle due strutture ha un aspetto importante nel modello.

Se le componenti spaziali e semantiche sono mappate con un grado di coerenza (ognuno su un asse), esse formano una coerenza spaziale (coherence space). Tutte le

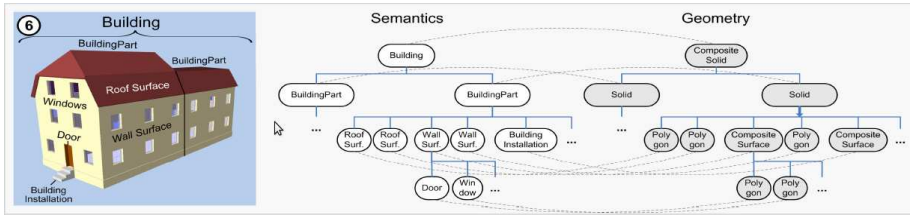

Figura 3.9: Caso 6

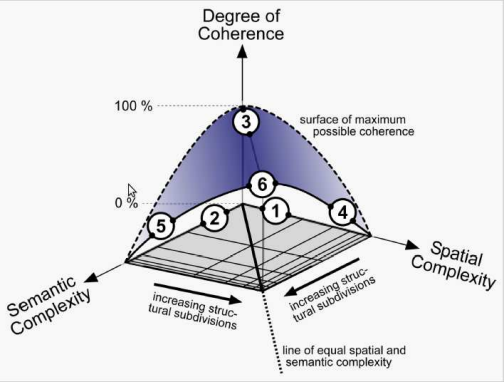

Figura 3.10: Possible degree of spatio-semantic coherence depending on spatial and semantic complexity.

istanze possono essere situate all'interno dello spazio rispetto alle loro caratteristiche, il tutto in Figura 3.10 dove illustra le interdipendenze tra geometria, semantica e la loro coerenza. La linea tratteggiata in fondo rappresenta un intervallo aperto dove la coerenza non è definita per l'inesistenza di semantica o di geometria. Le etichette 1-6 rappresentano le posizioni dei casi 1-6 sopra descritti.

Inoltre in tutti i casi riportati (ad eccezione per il caso 2) l'aspetto visivo del modello può essere identico, questo dimostra che, anche se i modelli sono simili possono differire sostanzialmente.

3.1.3 Modello Spaziale

Le proprietà spaziali degli oggetti CityGML sono descritte dal modello geometrico di GML3. Questo modello è basato sullo standard ISO 19107 Spatial Schema (Herreing 2001), che rappresenta le geometria 3D in accordo con la well-known Boundary Representation (B-Rep, cf. Foley et al. 1995). Attualmente CityGML usa solo un sottoinsieme delle geometrie definite in GML3.

Il modello geometrico consiste di geometrie primitive che possono essere combinate tra loro per formarne di complesse, composte o raggruppamenti. Per ogni dimensione c'è una sola geometria primitiva: un oggetto a zero dimensioni è modellato come un punto, uno a una dimensione come una curva, uno a due dimensioni come una superficie, infine uno a tre dimensioni come un solido. In CityGML una curva è limitata ad essere una linea diritta, per questo usata la sola classe *LineString* di GML3. Le superfici sono raffigurate da poligoni che definiscono una geometria piana.

Le combinazioni di geometrie possono essere: aggregati, geometrie complesse o composizioni di primitive.

In una aggregazione, *Aggregate*, le relazioni spaziali tra i componenti non sono restrittive, possono essere disgiunti, sovrapposti o disconnessi. GML3 mette a disposizione una aggregazione per ogni dimensione: *MultiPoint, MultiCurve, MultiSurface, MultiSolid*. Al contrario di una aggregazione, Complex è topologicamente struttura-

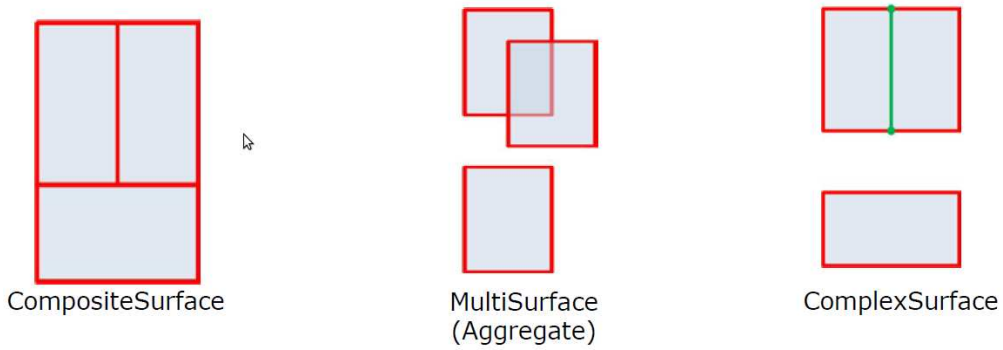

Figura 3.11: Diversi tipi di aggregazioni geometriche

ta: le sue parti devono essere disgiunte, non devono sovrapporsi ma è possibile che si tocchino al massimo lungo il perimetro. Composite è un tipo speciale di geometria complessa di GML3, può contenere solo elementi delle stesse dimensioni che devono essere disgiunti ma topologicamente connessi lungo il perimetro. Un Composite può essere un *CompositeSolid*, un *CompositeSurface* o un *CompositeCurve*.

Per il progetto in esame per la modellazione delle geometrie a due e tre dimensioni si è deciso di eseguire alcune semplificazioni. Tutte le geometrie basate su superfici sono memorizzate come poligoni, che sono aggregate in *MultiSurfaces*, *CompositeSurfaces*, *TriangulatedSurfaces*, *Solids*, *MultiSolids* come pure *CompositeSolids*. Questa semplificazione permette di sostituire la rappresentazione più complessa usata per queste classi GML3, con una più semplice in modo tale da usare solo una tabella nel database (SURFACE_GEOMETRY), come verrà spiegato in seguito.

Al fine di implementare la topologia, CityGML usa il concetto XML di XLinks fornito da GML. Ogni geometria che dovrebbe essere condivisa da diversi insiemi di geometrie, o alle diverse caratteristiche tematiche viene assegnato un identificativo univoco, che può essere referenziato da una proprietà di una geometria GML usando un attributo href. Uno svantaggio della topologia degli XLinks è dato dal fatto che la navigazione tra oggetti topologicamente connessi può essere fatta solo in una direzione, da un aggregazione al suo componente, e non bidirezionalmente come nel caso della topologia implementata in GML. Questa semplificazione è mostrata nella Figura 3.12

CityGML definisce anche le implicit geometry, che sono un miglioramento del modello geometrico di GML3.

Un *implicit geometry* è una geometria per cui la forma è memorizzata una sola volta come un prototipo, ad esempio un albero, un semaforo o un cartello stradale. Questi oggetti sono riusati o referenziati tutte le volte che la corrispondente caratteristica ricorre nel modello. Ogni occorrenza è rappresentata da un link alla geometria prototipo, da una matrice di trasformazione che viene moltiplicata per ogni coordinata 3D dell'oggetto e da un "anchor point" che indica il punto di base dell'oggetto nel sistema di coordinate di rifermento.

Questo tipo di rappresentazione rispetto alla modellazione esplicita, che usa un

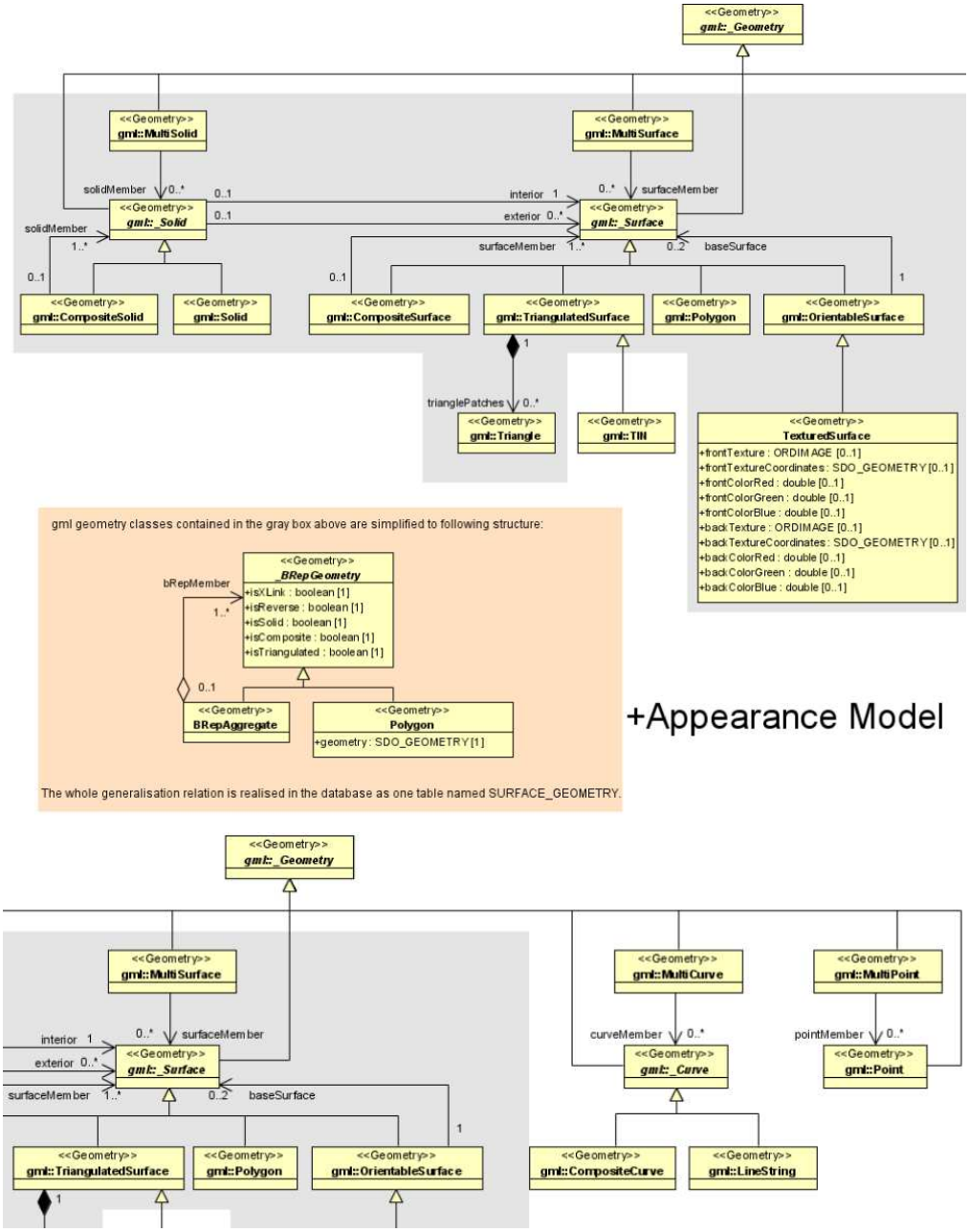

Figura 3.12: Modello Geometrico e Topologico

sistema di coordinante assoluto, presenta alcuni vantaggi: riduce lo spazio usato per la memorizzazione del modello, così da poter rendere possibile la memorizzazione nel modello di aree più grandi; la visualizzazione è accelerata; inoltre l'uso di versioni diverse di una forma è facilitato (ad esempio la rappresentazione di un albero in diverse stagioni), basta cambiare la libreria degli oggetti.

3.1.4 Modello Appearance

Contiene le informazioni relative all'aspetto delle superfici, le proprietà osservabili, è considerato una parte integrante della visualizzazione 3D dei modelli insieme alla semantica e alla geometria. Non rappresenta solo proprietà osservabili, ma tutte le categorie dette tema (theme) come gli infrarossi, inquinamento acustico o cedimenti strutturali dovuti a terremoti.

Ogni LOD può avere una sua *apperance* per uno specifico tema. Una *appearance* è composta da dati per ogni oggetto superficie (surface data). I temi sono rappresentati solo da un identificatore. Le *appearance* di un modello della città per un dato tema sono definite da un insieme di oggetti della classe Appearance, referenti il tema attraverso l'attributo theme.

Una proprietà costante è modellata come material, mentre una che dipende dalla posizione all'interno della superficie è modellata come texture. Ogni superficie può avere per un tema e un lato sia una texture sia un material. Se una superficie ha più material o più texture, ogni texture o material richiede un tema separato.

3.1.5 Modello Thematic

Il modello tematico definisce le classi per i più importanti tipi di oggetti all'interno del modello 3D. molte delle classi tematiche sono derivate dalle classi basi: Feature e FeatureCollection (definite dallo standard ISO 19109 e GML3). Le caratteristiche (features) contengono attributi spaziali e non spaziali, che sono mappati in corrispondenti tipi di dati e proprietà in GML3. Le proprietà sono rappresentate da associazioni con le classi geometriche. Il modello tematico comprende anche diversi tipi di relazioni tra le classi degli oggetti come aggregazioni, generalizzazioni e associazioni.

L'obiettivo della modellazione esplicita è quello di raggiungere un alto grado di interoperabilità semantica tra diverse applicazioni. Specificando i concetti di tematica e la loro semantica insieme con la loro mappatura a UML e GML3, diverse applicazioni possono fare affidamento su un ben definito insieme di tipi di oggetto, attributi e tipi di dati con una interpretazione standardizzata. In modo da rendere possibile anche lo scambio di oggetti e/o attributi che non son esplicitamente modellati in CityGML, sono stati introdotti i concetti di GenericCityObject e di GenericAttributes.

La classe a base di tutte le classi tematiche all'interno del modello CityGML è la classe CityObject, che fornisce una data di creazione e fine per la gestione della storia degli oggetti così come attributi generici e referenze esterne corrispondenti ad oggetti in dataset esterni. CityObject è una sottoclasse della classe Feature di GML, e in quanto tale eredita tutti i metadati e nomi della sua superclasse.

Le sottoclassi di CityObject comprendono diversi campi tematici del modello tematico della città: il terreno, mezzi di trasporto, vegetazione, bacini idrici e in particolare edifici. La classe CityFurniture è usata per rappresentare semafori, insegne, cartelli stradali, lampioni o oggetti simili. La classe GenericCityObject permette la modellazione di oggetti, che non sono esplicitamente definiti nello schema CityGML, come ad

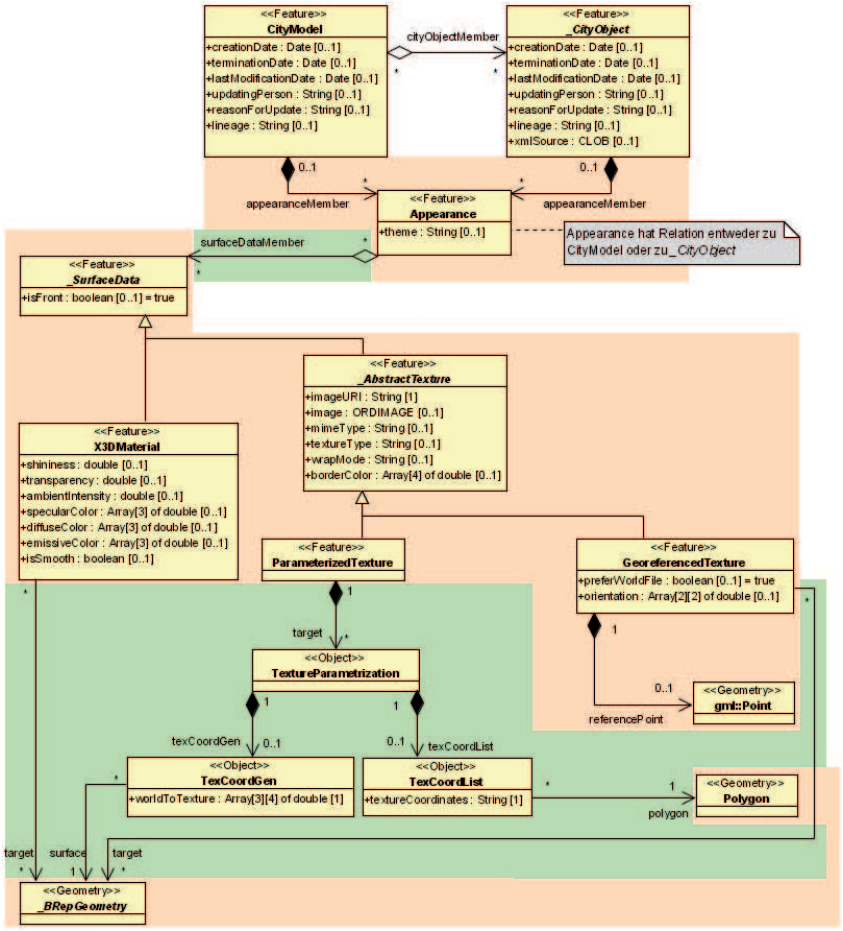

Figura 3.13: UML Appearance Module.

esempio ponti, sottopassi, tunnel che al momento non sono esplicitamente definite+i dallo standard.

CityGML modello Core Il modulo *CityGML Core* definisce i concetti e le componenti base dei dati in CityGML, quindi, è una dipendenza di tutti gli altri moduli. Il diagramma UML in Figura 3.14 illustra il *CityGML Core*. La classe base è la classe astratta _CityObject, fornisce la creazione e la scadenza dello storico delle caratteristiche, nonché la possibilità di modificare i riferimenti esterni allo stesso oggetto in altri database.

_CityObject è una sottoclasse di *Feature*, quindi eredita le proprietà della superclasse _GML. I CityObject possono disporre di molteplici nomi, che sono facoltativamente

qualificati da un codeSpace. Questo consente di distinguere tra, ad esempio, un nome ufficiale e un nome popolare o in diverse lingue.

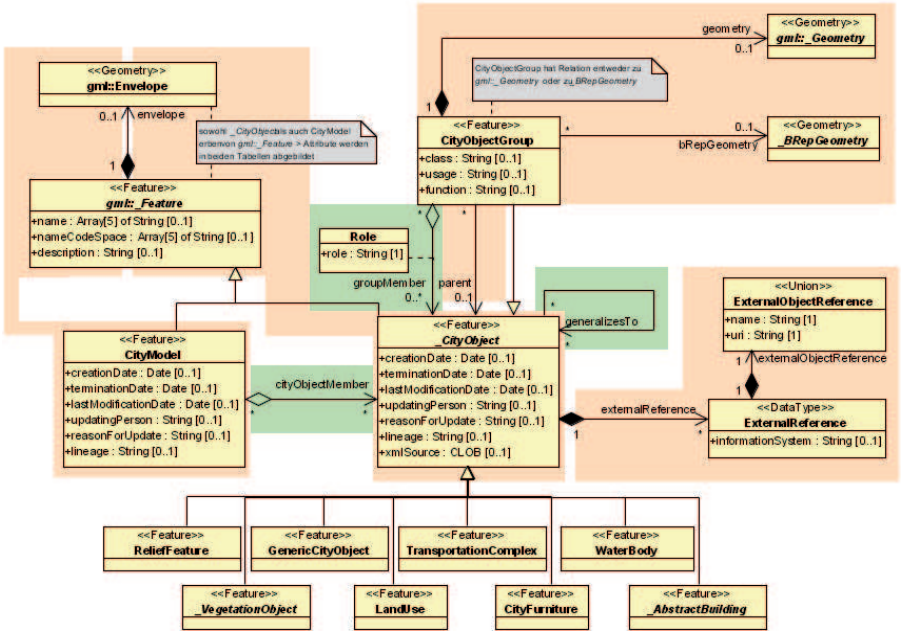

Figura 3.14: UML modello Core.

Modello Building Gli edifici possono essere rappresentati in quattro livelli di dettaglio (dal LOD1 al LOD4). CityGML permette la rappresentazione di edifici semplici formati da un solo componente così come quella di edifici più complessi composti da più componenti, ad esempio un edificio composto dalla casa principale, un garage e un portico. Le relazioni tra le parti degli edifici sono ricorsive, nel senso che una parte a sua volta può essere composta da più parti. Tutto questo è realizzabile attraverso le sottoclassi Building e BuildingPart di AbstractBuilding.

Nel caso di un immobile semplice, costituito da un solo pezzo, c'è un solo Building che eredita tutti gli attributi e le relazioni da AbstractBuilding. Comunque un edificio può essere costituito da più parti, BuildingParts, che ereditano tutte le proprietà da AbstractBuilding: class, function, usage, year of construction, year of demolition, roof type, measured height e il numero e l'altezza di ogni singolo piano. Inoltre, si possono assegnare indirizzi, Address, sia ai singoli edifici sia alle loro parti.

La geometria di un AbstractBuilding è realizzata attraverso una relazione con la classe GML_Composite. Per ogni livello di dettaglio è messa a disposizione una singola rappresentazione geometrica. Tutto ciò è messo in atto attraverso le relazioni lod1Geometry, ..., lod4Geometry. Infine, un singolo edificio può avere più rappresentazioni geometriche in diversi livelli di dettaglio allo stesso tempo.

Nel LOD1, il building model consiste nella rappresentazione geometrica del volume dell'edificio. Questa rappresentazione è ridefinita nel LOD2 da geometrie aggiuntive quali, MultiSurface e MulltiCurve, usate per modellare i dettagli architetturali quali tetti, colonne o antenne.

Nel LOD2 e in quelli superiori le facciate esterne di un edificio possono essere differenziate semanticamente con le classi BoundarySurface e BuildingInstallation. La prima rappresenta una parte dell'esterno dell'edificio con una funzione speciale come: muro (WallSurface), tetto (RoofSurface), suolo (GroundSurface) o superfici chiuse (ClosureSurface), come mostrato in figura Figura 3.15.

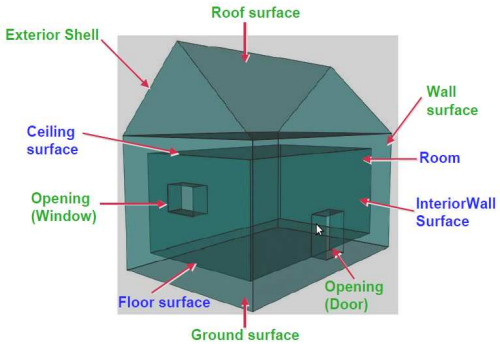

Figura 3.15: BoundarySurface

Nel LOD3, le aperture negli oggetti, come porte e finestre, possono essere rappresentati come oggetti tematici.

Infine, nel LOD4, anche gli interni di un edificio sono rappresentati nel modello attraverso la classe Room. Il raggruppamento di stanze è fatto in accordo ai criteri di raggruppamento definiti da CityGML. La superficie visibile di una stanza è geometricamente rappresentata come un solido o una multi-superficie. Semanticamente, BoundarySurface specifiche rappresentanti pavimenti (FloorSurface) soffitti (CeilingSurface) e muri interni essere (InteriorWallSurface). Infine, gli arredi interni, quali sedie, tavoli, etc. possono essere rappresentati con la classe BuildingFurniture.

Modello City furniture Gli oggetti che appartengono a questa classe sono oggetti immutabili come lanterne, semafori, segnaletica stradale, colonne, panchine, fermate dell'autobus.

Gli oggetti facenti parte di questa classe possono avere un ruolo importante in applicazioni per la simulazione di eventi quali, per esempio, il traffico cittadino.

Questa classe può avere gli attributi: class e function. L'attributo class permette una classificazione dell'oggetto e può apparire una sola volta.

L'attributo function descrive l'area tematica a cui appartiene l'oggetto (ad esempio trasporti, regolazione del traffico,etc.) e può comparire più volte.

Gli elementi di arredo urbano possono essere rappresentati nel modello con la loro geometria oppure, come avviene nel maggior numero di casi, gli oggetti dello stesso

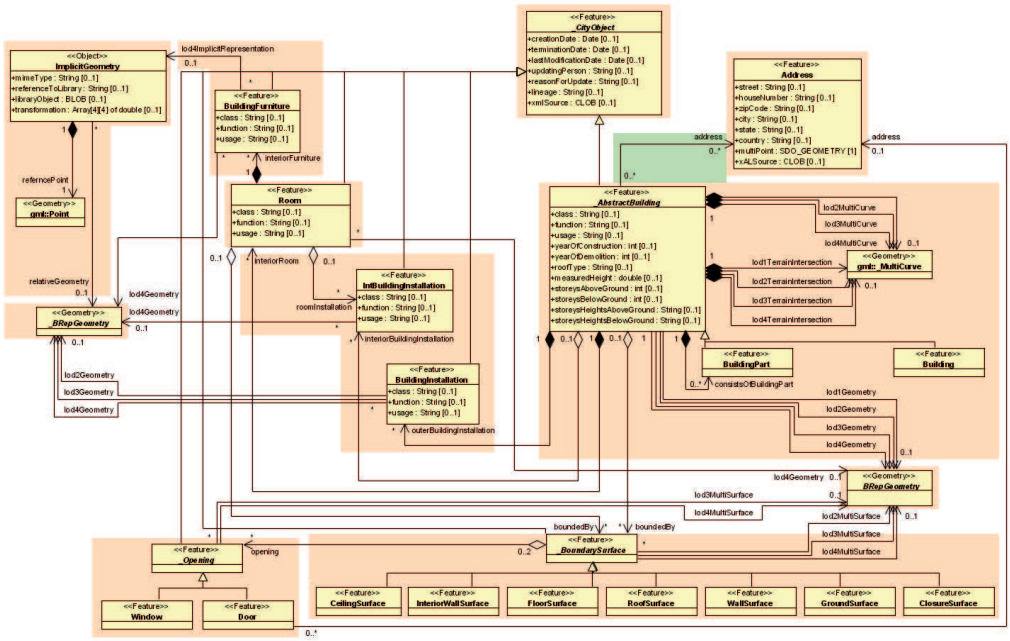

Figura 3.16: UML Building Module.

tipo sono rappresentati con la stessa geometria, prototipi. Nel secondo caso si tiene in memoria una sola copia della geometria nelle coordinate del sistema di riferimento locale, e poi viene referenziato da tutti gli oggetti che hanno quella particolare forma.

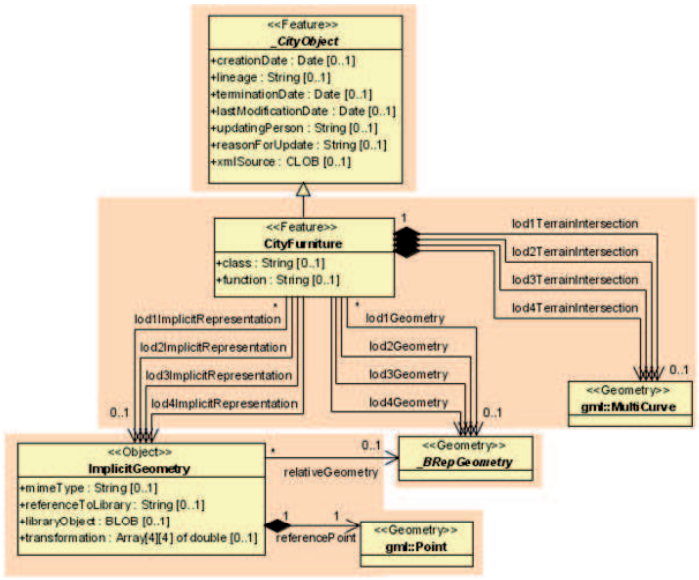

Figura 3.17: UML modello Furniture

Modello Digital terrain Model(DTM) Una parte necessaria del modello della città è il terreno. In CityGML il terreno è rappresentato dalla classe ReliefFeautre, un elemento appartenete a questa classe consiste di una o più entità della classe ReliefComponent, che sono entrambi derivati di CityObject, pertanto ne ereditano tutti gli attributi e relazioni.

Alla classe ReliefFeautre sono associati diversi concetti di rappresentazione del terreno che possono coesistere. Il terreno può essere specificato come un raster o grid (RasterRelief), come TIN (Traingulated Irregular Network, TINRelief), da break lines (BreaklineRelief) o, infine, da mass point (MasspointRelief). Tutti e quattro i tipi sono implementati dalle corrispondenti classi di GML3 rispettivamente: RectifiedGridCoverage, TriangulatedSurface o TINs, Curves, Points.

Nel caso di TriangulatedSurface, i triangoli sono dati esplicitamente, mentre nel caso di GML3 TINs sono rappresentati solo i punti 3D, le triangolazioni sono ricostruite attraverso metodi standard (ad esempio con le triangolazioni di Delaunay). Break Lines sono rappresentate da curve 3D, e i Mass point semplicemente con un insieme di punti 3D.

In CityGML i quattro tipi di suolo possono essere combinati tra loro in modo diverso, permettendo così un alto grado di flessibilità. Prima di tutto ogni tipo può essere rappresentato nei cinque livelli di dettaglio, rispecchiando accuratezza e risoluzione diverse. Come seconda cosa, una parte del terreno può essere descritta con la combinazione di diversi tipi di rappresentazione. Infine, regioni vicine possono essere rappresentate da diversi modelli di terreno. Per facilitare questa combinazione, ogni oggetto rappresentante il terreno è corredato di un attributo spaziale che ne denota l'estensione della validità.[*extent of validity*] (Figura refvalidity). Nella maggior parte dei casi, la misura di validità di un dataset raster regolare corrisponde al suo bounding box. Questa estensione di validità è rappresentata da un poligono 2D, che può avere dei buchi. Questo concetto consente, ad esempio, la modellazione di un terreno da una coarse grid, dove sono rappresentate alcune regioni di dettaglio maggiore, TIN ad alta precisione.

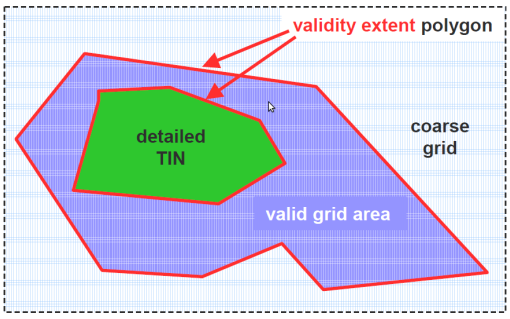

Figura 3.18: Validità extent poligoni

L'accuratezza e la risoluzione del DTM non dipende, necessariamente, da quella del modello degli edifici. Infatti, c'è la possibilità di integrare un modello degli edifici a un livello di dettaglio maggiore con un DTM a un livello di dettagli inferiore.

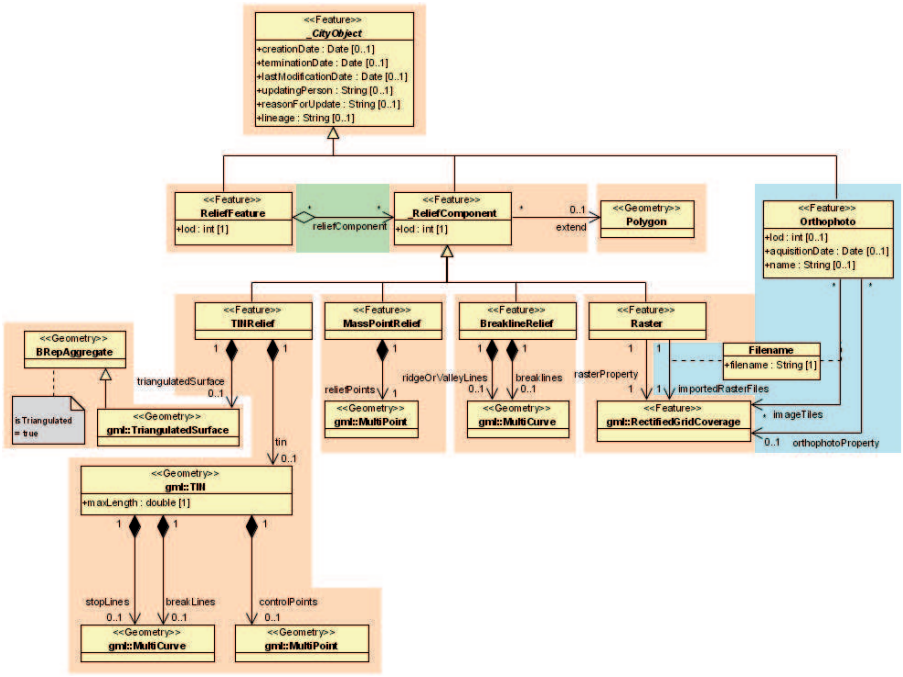

Figura 3.19: UML modello DTM.

Modello Generic CityObject Il concetto di oggetti generici e attributi è stato introdotto per assicurare la memorizzazione e lo scambio di oggetti 3D, che non sono modellati esplicitamente da classi di CityGML o che richiedono attributi addizionali.

Queste estensioni sono realizzate attraverso le classi GenericCityObject e GenericAttribute.

Un'istanza della classe GenericCityObject può avere gli attributi: class, function e usage. Il primo rende possibile una classificazione dell'oggetto all'interno dell'area tematica come: ponte, tunnel, linea elettrica, diga o non classificato.

Il secondo attributo, descrive a quale area tematica l'oggetto generico appartiene, ad esempio trasporti, architetturale, rifornimento di energia elettrica o di acqua, etc. Infine, l'attributo usage, può essere usato se l'oggetto attualmente è usato in modo diverso da quanto specificato dall'attributo function.

Tutti i CityObject possono avere un numero arbitrario di GenericAttributes.

I tipi di dato possono essere: stringhe, interi, numeri a virgola mobile, URI, date, BLOB,_BRepGeometry o qualsiasi altro tipo di geometria. L'attributo type è definito dalla selezione della sottoclasse specifica.

La geometria di un GenericCityobject può essere sia una geometria esplicita di GML3 sia una ImplicitGeometry. Nel primo caso, l'oggetto può avere solo una geometria per ogni livello di dettaglio, che può essere un qualsiasi tipo di geometria 3D definita da GML. Nel secondo caso sono forniti un anchor point e una matrice di trasformazione per l'oggetto.

Per specificare 3-dimensionale l'esatta intersezione dell'oggetto generico, del DTM con quest'ultimo la geometria può avere una TerrainIntersectionCurve per ogni LOD.

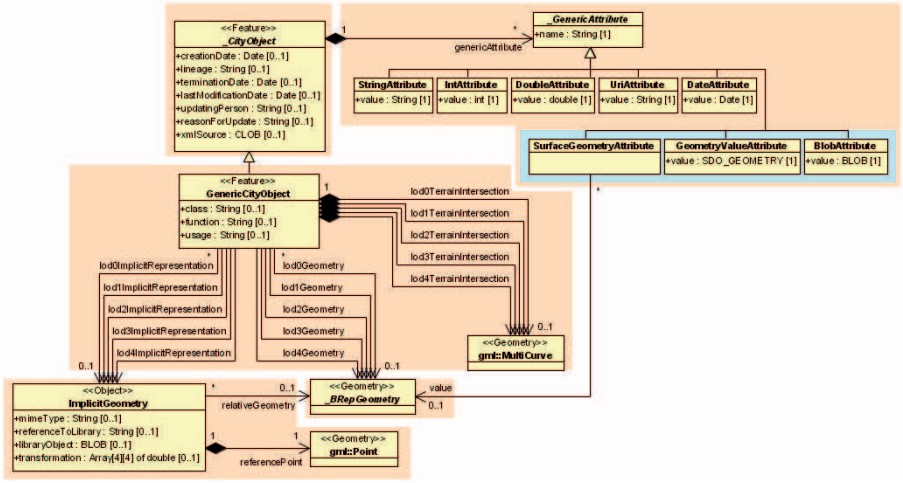

Figura 3.20: UML modello Generic CityObjec.

Modello Land use Gli oggetti appartenenti a questa classe descrivono l'uso che si fa di quella parte di superficie terrestre.

Ogni oggetto facente parte della classe LandUse può avere gli attributi class usato per classificare l'oggetto (ad esempio: area industriale, fattoria,etc,), function usato per definire lo scopo dell'oggetto ed infine usage che può essere usato per mettere in evidenza che l'oggetto è usato, o svolge una funzione diversa da quella definita nel campo function. Il primo attributo può essere usato una sola volta, mentre gli altri due possono comparire più volte.

Un oggetto LandUse è definito per ogni LOD e può avere una geometria diversa per ogni LOD. La sua superficie deve avere coordinate 3-dimensionali e deve appartenere alla classe MultiSurface di GML3.

Modello Transportation Il modello dei trasporti di CityGML è un modello multi-funzionale, multi-scala che mette a fuoco sia gli aspetti topologici e funzionali sia quelli geometrici/topologici.

I componenti di questa classe sono rappresentati come un grafo nel 'LoD 0', a partire dal 'LoD 1' tutti gli oggetti sono descritti geometricamente da superfici 3D.

La classe principale è TransportationComplex, che rappresenta, per esempio, una strada, un binario, o una piazza. É composta da due parti: TrafficArea e AuxiliaryTrafficArea.

La figura sotto mostra un esempio di TransportatioComplex nel 'LoD 2' all'interno di un modello 3D. La strada, Road, consiste di diverse TrafficAreas per il passaggio pedonale, strade, parcheggi, e AuxiliaryTrafficArea, le fioriere.

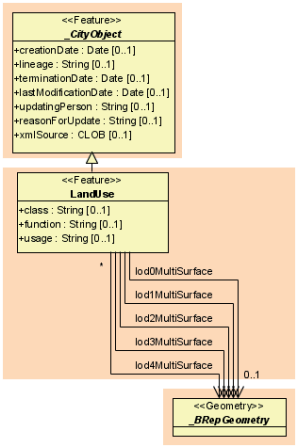

Figura 3.21: UML modello Land Use.

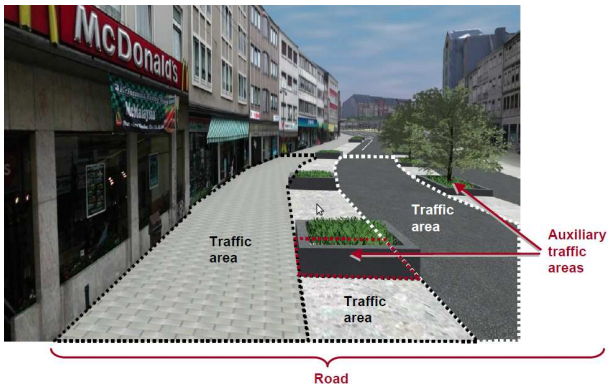

Figura 3.22: Esempio in ‘LOD 2’ di TransportatioComplex.

La strada è rappresentata come TransportationComplex, che a sua volta è suddivisa in TrafficAreas e AuxiliaryTrafficAreas. Gli elementi appartenenti alla classe TrafficArea, sono quegli elementi ritenuti importanti per l’uso delle diverse aree, come marciapiedi, piste ciclabili o carreggiate. Mentre AuxiliaryTrafficAreas descrive atri elementi della strada, quali aree verdi o bordi dei marciapiedi.

Gli oggetti della classe TransportationComplex possono essere differenziati tematicamente usando le sottoclassi Track, Road, Railway e Square. Ogni elemento ha gli attributi function, che descrive lo scopo dell’oggetto (ad esempio: strada statale, aeroporto) e usage, referenti la lista esterna.

Inoltre, ogni TrafficArea può avere gli attributi function che descrivono se l’oggetto è un’area pedonale o una pista ciclabile, usage al contrario di function indica quale tipo di mezzo di trasporto può usare l’oggetto, ed infine surfaceMaterial che specifica il tipo di pavimentazione e può essere usato anche per la classe AuxiliaryTrafficArea. TransportationComplex è una sottoclasse di CityObject.

La rappresentazione di un oggetto di questa classe varia a seconda del diverso livello di dettaglio. Come detto in precedenza al livello di dettaglio 0 si usano degli oggetti lineari in modo da stabilire una rete lineare. A partire dal livello di dettaglio 1 ogni oggetto ha una geometria esplicita rappresentante la forma dell'oggetto. Nei livelli di dettaglio 2, 3 e 4 gli oggetti sono suddivisi anche tematicamente in TrafficAreas, che sono usate dai mezzi di trasporto, come automobili, treni, tram, aerei, biciclette o pedoni e in AuxiliaryTrafficAreas, che sono di minore importanza per il trasporto, ad esempio le aiuole.

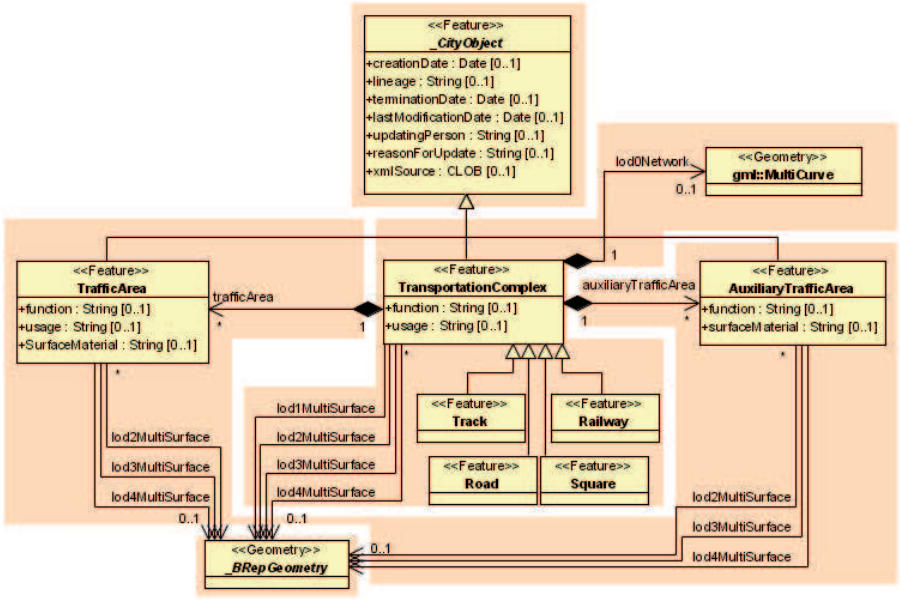

Figura 3.23: UML modello Transportation.

Modello Vegetation Il modello della vegetazione di CityGML distingue tra singoli elementi quali alberi, ed elementi composti quali aree verdi e foreste. Gli oggetti del primo tipo sono modellati attraverso la classe SolitaryVegetationObject, mentre per quelli del secondo tipo si usa la classe PlantCover.

La rappresentazione geometrica di una piantagione può essere fatta usando le classi *MultiSurface* o *MultiSolid*, a seconda dell'estensione in altezza delle piante. Ad esempio per una foresta è più appropriato usare *MultiSolid*, come si vede in figura, mentre per un prato una *MultiSurface*.

Un oggetto di tipo SolitaryVegetation può avere gli attributi: class (ad esempio: albero, cespuglio), species che indica il nome della specie, function, height, trunkDiameter e appartenente alla classe PlantCover esempio: foresta), function (ad crownDiameter.

Un elemento può avere gli attributi: class (ad esempio: parco nazionale), averageHeight.

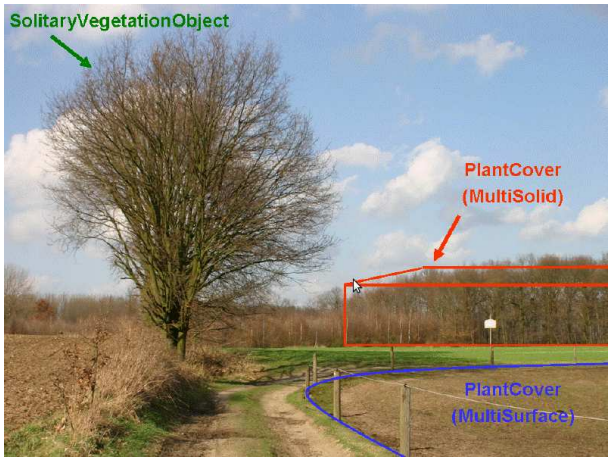

Figura 3.24: Esempio di modello di vegetazione.

Fintanto che sia SolitaryVegetation sia PlantCover sono CityObject, ereditano tutti gli attributi degli oggetti, in particolare il nome (gml:name) e le ExternalReference agli oggetti corrispondenti.

La geometria delle singole piante può essere definita nei 'LOD 1-4' con coordinate assolute o con prototipi con l'uso di ImplicitGeometry, a questi ultimi si possono applicare anche appearance diverse, per esempio relative alle stagioni. Come per tutti gli altri oggetti anche quelli appartenenti a queste classi, possono avere una diversa rappresentazione in ogni livello di dettaglio. Mentre un SolitaryVegetationObject è associato alla classe _Geometry, che rappresenta una geometria qualsiasi definita in GML (attraverso la relazione lodXGeometry), un PlantCover è limitato ad essere un **MultiSolid** o un **MultiSurface**.

Modello WaterBody Il modello delle acque rappresenta l'aspetto tematico e la forma 3D di rive, canali, laghi e bacini idrici. Nei LOD2-4 le acque sono circondate da superfici tematiche distinte. Queste superfici sono obbligatoriamente appartenenti alla classe WaterSurface, definita come delimitazione tra l'aria e l'acqua, mentre la classe, opzionale, WaterGraoundSurface è definita come confine tra l'acqua e il suolo sottostante, infine, possono avere zero o più WaterClosureSurfaces, definite come confini virtuali tra diversi oggetti di tipo WaterBodies o tra l'acqua e la fine della regione modellata (Figura Figura 3.26).

Ogni elemento di questo gruppo può avere gli attributi: class (esempio: lago, riva, fiume), function (ad esempio: piscina comunale) ed infine usage (ad esempio: navigabile) referente una lista esterna.

Come tutte le altre classi analizzate finora, anche WaterBody è una sottoclasse di CityObject. La rappresentazione geometrica di un oggetto di questo tipo varia a seconda del livello di dettaglio. Sia nel LOD0 che nel LOD1 le rive sono modellate

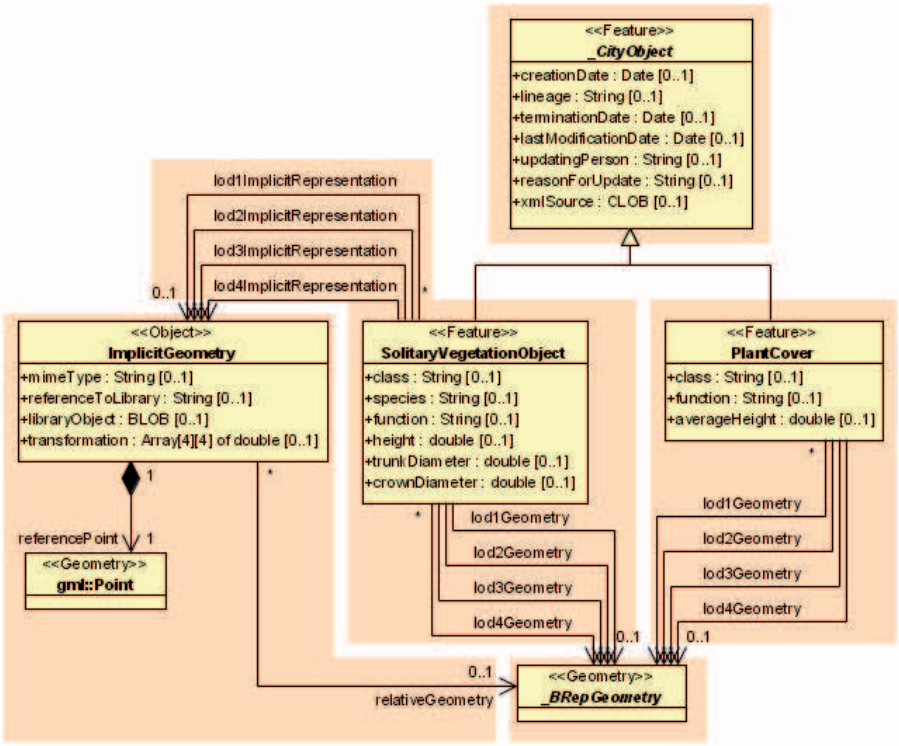

Figura 3.25: UML Modello Vegetation.

come MultiCurve, laghi, mari e oceani di significative dimensioni sono modellati come MultiSurfaces.

A partire dal LOD1, le acque possono essere rappresentate anche come solidi volumetrici, rappresentati attraverso la classe Solid. Se un bacino idrico è rappresentato come un solido nel LOD2 o in LOD superiori, la superficie tematica WaterGroundSurface e/o corrispondente, WaterSurface WaterClosureSurface, devono corrispondere con il perimetro esterno del solido. Il modello delle acque include implicitamente il

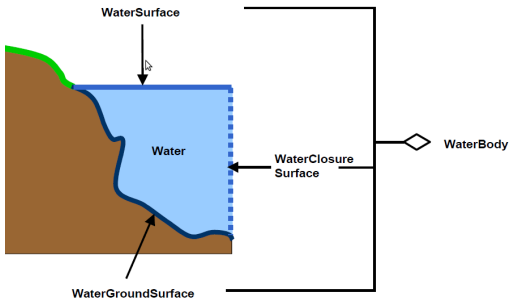

Figura 3.26: Illustrazione di un WaterBody definito in CityGML.

concetto di TerrainIntersectionCurve (TIC), per esempio, per specificare l'esatta intersezione del DTM con la geometria 3-dimensionale di un WaterBody o per adattare un WaterBody o una WaterSurface al DTM circostante.

Il perimetro che definisce la superficie del poligono delinea implicitamente l'intersezione dell'oggetto "acqua" con il terreno.

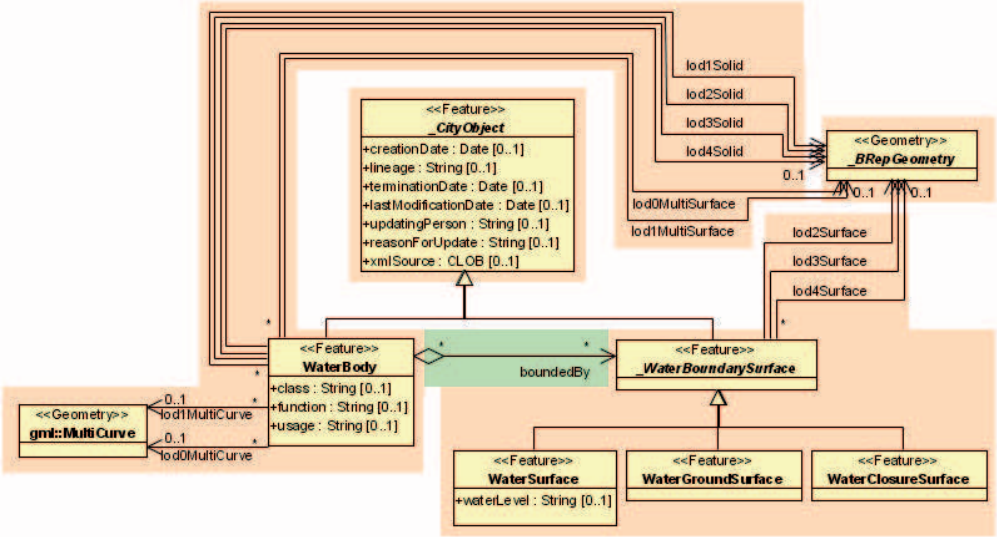

Figura 3.27: UML modello WaterBody.

3.2 OGC KML

KML (Keyhole Markup Language) è una grammatica XML usata per codificare e trasportare rappresentazioni di dati geografici per visualizzarli in un browser earth, come un globo virtuale 3D (Google Earth), applicazioni 2D per web browser o applicazioni 2D mobili. La visualizzazione geografica non include la presentazione di dati geografici nel globo, ma anche il controllo della navigazione degli utenti nel senso di dove andare e dove guardare. La parola Keyhole deriva dal nome del software progenitore di Google Earth; questo fu prodotto a sua volta dalla Keyhole Inc., che fu acquisita da Google nel 2004. Il termine keyhole onora il nome dei satelliti KH, utilizzati nel vecchio sistema di ricognizione militare statunitense. Google(membro OGC) ha presentato KML in modo tale da essere evoluto all'interno del processo di approvazione OGC. La versione 2.2 è uno standard adottato e attuato da OGC. Le future versioni potrebbero essere armonizzate con i principali standard OGC che compongono la linea di base dello standard OGC. Attualmente la versione 2.2 KML utilizza una serie di elementi di geometria derivati da GML 2.1.2. Questi elementi includono *point*, *lineString*, *linearRing* e *polygon*.

KML può essere usato per:

- annotare la Terra
- Specificare le icone e le etichette per identificare la posizione sulla superficie del pianeta
- Creare differenti posizioni di ripresa per definire una vista unica per le feature KML
- Definire le sovrapposizioni di immagini da allegare al suolo o sullo schermo.
- Definire gli stili per specificare le appearance delle feature KML.
- Scrivere descrizioni HTML di feature KML, includendo collegamenti ipertestuali e immagini.
- Organizzare feature KML in gerarchie.
- Individuare, aggiornare e recuperare documenti XML da percorsi di rete locale o remota.
- Definire la posizione e l'orientamento delle texture degli oggetti 3D.

3.2.1 Struttura KML

Un file KML è processato nella stessa modalità con la quale i web browser processano i file HTML e XML; da qui il nome di Browser Earth , utilizzato per riferirsi ai programmi che visualizzano i file KML. Il linguaggio KML ha infatti una struttura basata su tag, con nomi e attributi che consentono di definire specifiche caratteristiche di visualizzazione.

La gerarchia degli elementi del linguaggio KML e mostrata nella Figura 7.27.

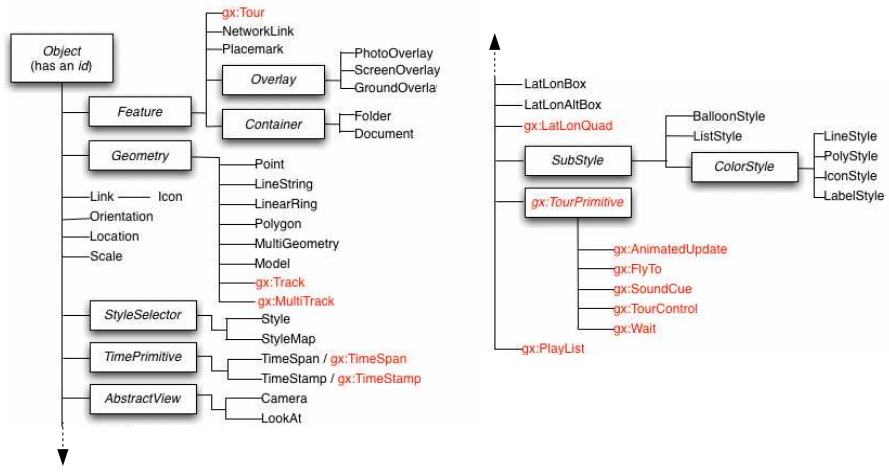

Figura 3.28: Gerarchia elementi linguaggio KML

Si noti che tutti gli elementi astratti(indicati nel diagramma all'interno delle caselle) non sono attualmente utilizzati nei file KML. Sono utilizzati a livello logico per poter aggregare in maniera più comprensibile e immediata gli elementi ad essi derivati. Tutti gli elementi di KML derivano da Object, dal quale ereditano l'attributo *id*, che permette l'identificazione univoca di ogni oggetto.

3.2.2 Feature

L'elemento astratto *Feature* è il genitore degli elementi *NetworkLink* (riferimenti a file KML o KMZ remoti o locali), *Placemark* (segnalibri geografici), *Overlay* (il progenitore di ScreenOverlay GroundOverlay sovrappone immagini applicabili allo schermo o al terreno) e Container (Elemento astratto che può contenere uno o più elementi *Feature*e consente la generazione di gerarchie innestate)

L'elemento Feature fornisce un insieme di strumenti per la descrizione delle informazioni tematiche relative ad un oggetto; esso contiene gli attributi che definiscono il nome <Name>, l'indirizzo <Address> e <AddressDetails>, il numero di telefono associato <PhoneNumber>, la descrizione dell'oggetto in forma breve <Snippet>, e <Description> che definisce una descrizione più articolata sia in forma testuale che attraverso informazioni HTML. Non sono supportate tecnologie come PHP,JSP o ASP. vedere anche style e atom.

3.2.3 TimePrimitive

Una possibilità offerta da KML è la presentazione del tempo in cui si tiene la rappresentazione di un oggetto; qualsiasi elemento derivato da Feature, quindi <Placemark>, <Conteiner>, <Overlay>, e <NetworkLink> possono contenere un elemento di tipo TimePrimitive, che rappresenta l'astrazione per la resa dell'intervallo temporale <TimeSpan> o dell'istante <TimeStamp> al quale risale l'osservazione dell'oggetto.

3.2.4 Region

Le Regioni sono una potente funzione di KML che consente di aggiungere grandi insiemi di dati a Google Earth senza sacrificare le prestazioni. I dati vengono caricati e prelevati solo se si trovano alla vista dell'utente e se occupano una certa porzione dello schermo. <LatLonAltBox> individua la zona di interesse, e un oggetto <Lod>, il quale determina il livello di dettaglio della rappresentazione, cioè quale dimensione debba avere la regione proiettata sullo schermo per essere visibile e il grado massimo e minimo di dissolvenza, in termini di variazione massima e minima del numero di pixel, che essa adotta quando il punto di vista si avvicina o si allontana dall'area risultando più sfocata a grande distanza e via via più definita quando ci si avvicina. Nella Figura 3.29 vediamo sia i parametri per la LatLonAltBox e per la definizione del livello di dettaglio.

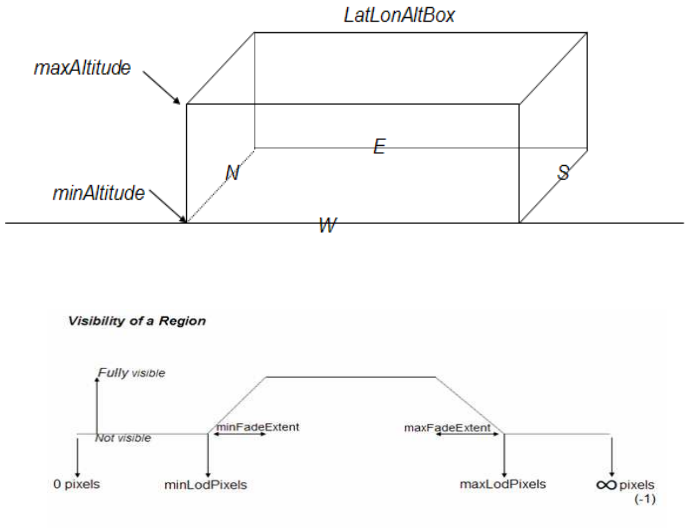

Figura 3.29: LatLonAltBox e LoD

3.2.5 Abstract Wiev

L'elemento astratto Abstract Wiev è esteso dagli elementi <LookAt> e <Camera>. <LookAt> è un elemento molto importante del linguaggio, in quanto definisce la vista dell'oggetto, e viene associato ad ogni elemento figlio di Feature. Esso raccoglie, oltre alle informazioni sulle coordinate geografiche dell'oggetto, anche dati che possano rendere la rappresentazione più specifica, quali l'inclinazione del punto di vista rispetto all'oggetto <tilt>, la sua distanza da esso <range> e l'angolo della visuale rispetto all'orizzonte <heading> Figura 3.30.

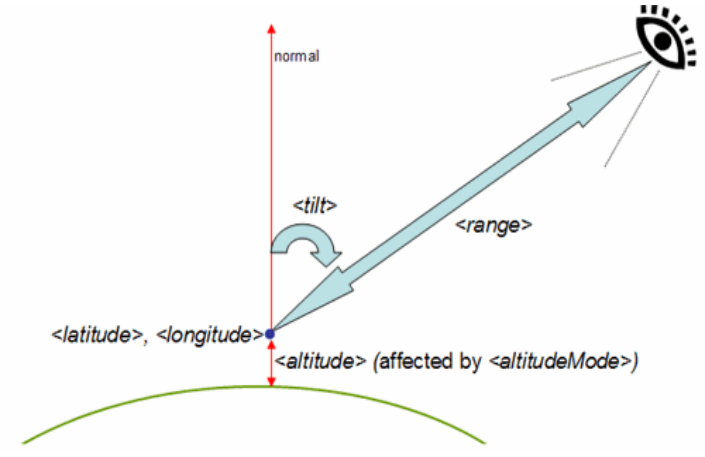

Figura 3.30: parametri di visualizzazione

L'elemento <camera> rappresenta la telecamera virtuale che individua la scena. Definisce la posizione della telecamera rispetto la superficie della terra. La posizione della camera è definita da <longitude>, <latitude>, <altitude>, e sia <altitudeMode>, La direzione di visione della camera è definita da <heading>, <tilt> e <roll>. Può essere un elemento figlio di una Feature o di <NetworkLinkControl>

3.2.6 Geometry

Gli elementi del modello geometrico in KML hanno tutti come elemento padre la classe astratta Geometry. A partire da essa, sono definiti gli elementi che fanno parte della geometria: <Point> (punto), <LineString> (insieme di una o più linee), <LinearRing> (anello), <Polygon> (superficie), <MultiGeometry> (geometria composta) e <Model> (modello tridimensionale). Le primitive su cui si basa il modello geometrico di KML sono tre, corrispondenti al punto per la dimensione 0, alla linea per la dimensione 1 e al poligono per la dimensione 2. KML non definisce invece una vera e propria primitiva geometrica per la terza dimensione. All'interno di un file KML, ogni oggetto è obbligatoriamente associato ad una locazione, individuata da una longitudine e ad una latitudine, ed opzionalmente da una altitudine. Il formato delle coordinate è dettato dal sistema di coordinate di riferimento WGS84, l'ultima revisione del WGS (World Geodetic System), il sistema che definisce un frame di riferimento fisso per la Terra, utilizzato in geodesia e in navigazione. In KML, questo è l'unico sistema di riferimento adottato; non è infatti possibile specificare un arbitrario sistema per le coordinate. Gli oggetti della geometria sono collocati in un file KML all'interno di un elemento <Placemark> o di un elemento <MultiGeometry>. Quest'ultimo, a sua volta, può esistere all'interno di un <Placemark> o di un'altra geometria composta, creando eventualmente una gerarchia di geometrie innestate l'una nell'altra. L'elemento <Placemark> invece è definito come Feature con geometria associata e rappresenta un segnalibro geografico, cioè uno strumento per visualizzare e memorizzare una luogo o la posizione di un oggetto. Un segnalibro geografico può contenere al massimo un oggetto di tipo Geometry; ciò significa che un <Placemark> può anche non visualizzare alcun oggetto, ma non esiste la possibilità di uno stesso segnalibro contenga geometrie che appartengano a gerarchie diverse, ovvero che non siano raggruppate in un elemento <MultiGeometry> comune. Un esempio di Placemark, associato in questo caso ad un punto, è il seguente:

```
<Placemark>
  <name>Placemark</name>
  <description>Esempio di Placemark</description>
  <LookAt>
    <longitude>-90.868798</longitude>
    <latitude>48.2533038</latitude>
    <range>440.8</range>
    <tilt>8.3</tilt>
    <heading>2.7</heading>
  </LookAt>
```

```
  <Point>
    <coordinates>-90.869489,48.254500,0</coordinates>
  </Point>
</Placemark>
```

Punto In KML, un punto definisce una posizione geografica, ed è espresso comunicandone longitudine, latitudine e, opzionalmente, altitudine. L'ordine dei tre valori è importante e deve essere mantenuto. Tutte questi parametri sono rappresentati da numeri a virgola mobile; mentre il primo e il secondo sono espressi in gradi e sono soggetti a restrizioni sui valori che possono assumere, l'altitudine non ha limitazioni, in quanto esprime l'altezza del punto, in metri sopra livello del mare. La longitudine rappresenta la distanza angolare dal meridiano fondamentale e può assumere valori compresi tra -180 e 180; la latitudine, invece, è l'angolo che la verticale di un punto sulla superficie della Terra forma con il piano equatoriale e non può assumere valori maggiori di 90 gradi e minori di -90 gradi. É qui riportato un semplice esempio di punto, di longitudine -90.869489, altitudine 48.254500 e altitudine 10.

Un punto può racchiudere, oltre all'informazione sulle coordinate, altri dati, non obbligatori, che specificano se esso sia connesso al terreno, se la linea o il percorso di cui fa eventualmente parte debba seguire il terreno, e quale sia la sua modalità di altitudine. Di seguito si mostra un esempio di punto, completo di tutte le informazioni addizionali:

```
<Point id="ID">
  <extrude>0</extrude>
  <tessellate>0</tessellate>
  <altitudeMode>clampToGround</altitudeMode>
  <coordinates>-90.869489,48.254500</coordinates>
</Point>
```

Il punto descritto dall'esempio è sul livello del mare, a longitudine -90.869489 e latitudine 48.254500, in quanto la sua altitudine non è specificata, e sarebbe anzi ignorata, qualora lo fosse: il tag <altitudeMode> con valore clampToGround indica infatti che il valore di altitudine non è significativo e non è quindi da considerare. AltitudeMode definisce altri due tipi di comportamento: quello per cui l'altitudine viene riferita all'elevazione del terreno della posizione a cui il punto si trova relativeToGround, e quello per il quale l'altitudine è assoluta, e quindi riferita al livello del mare absolute. I due flag extrude e tessellate, infine, indicano rispettivamente che il punto non è connesso al terreno, anche qualora facesse parte di un percorso.

Rappresentare una linea KML distingue due tipi di linea: quella chiusa e quella aperta. La prima, rappresentata dall'elemento <LineString>, può consistere in una singola linea oppure in un insieme connesso di segmenti, ognuno dei quali è individuato da una coppia di punti: quello iniziale e quello finale. Qui sotto è riportato un esempio di documento KML che contiene un Placemark (inserito all'interno del contenitore radice Document) al quale è associata una singola linea, con punto iniziale (-122.364383,

37.824664, 50) e punto finale (-122.364152, 37.824322, 50). In questo caso, quindi, la linea è individuata da esattamente due tuple di coordinate, quella per il primo punto e quella per il secondo punto che descrive il segmento.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.1">
<Document>
  <name>LineString.kml</name>
  <open>1</open>
  <LookAt>
    <longitude>-122.36415</longitude>
    <latitude>37.824553</latitude>
    <altitude>0</altitude>
    <range>150</range>
    <tilt>50</tilt>
    <heading>0</heading>
  </LookAt>
  <Placemark>
    <name>unextruded</name>
    <LineString>
      <extrude>1</extrude>
      <tessellate>1</tessellate>
      <coordinates>
        -122.364383,37.824664,50
        -122.364152,37.824322,50
      </coordinates>
    </LineString>
  </Placemark>
</Document>
</km>
```

Nel caso in cui la linea sia composta, per descriverla KML utilizza una lista ordinata di punti, che rappresenta il percorso che individua la linea. Ciascun punto, quindi, eccetto il primo e l'ultimo, è punto finale di un segmento, ed iniziale del segmento successivo.

Un LinearRing è invece un anello, cioè un insieme connesso di segmenti chiuso, in cui il primo punto del primo segmento ed ultimo dell'ultimo segmento coincidono. Pertanto, l'anello contiene obbligatoriamente almeno 4 tuple di coordinate, con la prima e l'ultima uguali, a disegnare il segmento di chiusura

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.1">
<Placemark>
  <name>LinearRing.kml</name>
  <LinearRing>
    <coordinates>
        -122.365662,37.826988,0
        -122.365202,37.826302,0
        -122.364581,37.82655,0
```

```
        -122.365038,37.827237,0
        -122.365662,37.826988,0
      </coordinates>
    </LinearRing>
  </Placemark>
</kml>
```

Sia l'elemento <LineString> che l'elemento <LinearRing> possono avere, oltre agli attributi derivati da Object e alle coordinate obbligatorie, le informazioni su estrusione (<extrude>, <tessellate>) e la modalità di altitudine (<altitudeMode>).

Rappresentare una superficie In KML, le superfici sono tutte poligonali. Non esistono pertanto superfici dal perimetro individuato da una curva (cerchi, ellissi, etc.). L'elemento che rappresenta la superficie è infatti un poligono <Polygon>, che viene definito obbligatoriamente da un perimetro esterno <outerBoundaryIs> e può possedere facoltativamente dei perimetri interni <innerBoundaryIs>. Sia il perimetro esterno che quello interno sono anelli (LinearRing), ovvero linee composte chiuse. L'esempio rappresenta un pentagono con un confine interno, sempre di forma pentagonale.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.1">
<Document>
  <name>Polygon.kml</name>
  <open>1</open>
    <Placemark>
    <name>Pentagono bucato</name>
    <Polygon>
      <altitudeMode>relativeToGround</altitudeMode>
      <outerBoundaryIs>
        <LinearRing>
          <coordinates>
            -122.366278,37.818844,30
            -122.365248,37.819267,30
            -122.365640,37.819861,30
            -122.366669,37.819429,30
            -122.366278,37.8188\44,30
          </coordinates>
        </LinearRing>
      </outerBoundaryIs>
      <innerBoundaryIs>
        <LinearRing>
          <coordinates>
          -122.366212,37.818977,30
          -122.365424,37.819294,30
          -122.365704,37.819731,30
          -122.366488,37.819402,30
          -122.366212,37.818977,30
          </coordinates>
        </LinearRing>
```

```
      </innerBoundaryIs>
    </Polygon>
  </Placemark>
</Document>
</kml>
```

Come le altre entità del modello geometrico di KML, anche i Polygon possono essere estrusi con gli attributi <extrude> e <tessellate>, e avere una <altitudeMode> settata.

Rappresentare un solido KML non definisce un elemento che modelli esplicitamente i solidi; ci sono quindi due possibili alternative per descrivere una entità di questo tipo: l'utilizzo dell'estrusione o la scomposizione del solido nelle facce che ne costituiscono il confine. Comunque, qualsiasi sia il metodo utilizzato, il linguaggio KML non fornisce il supporto per il disegno di solidi quali sfere, tori, cilindri e coni, essendo possibile rappresentare solo le superfici poligonali. Il primo metodo consiste nel disegnare solo il Polygon che definisce la faccia superiore di un solido e, tramite il meccanismo di estrusione applicato ad essa, tracciare la perpendicolare al terreno per ogni vertice della superficie. Infatti, quando un poligono viene estruso, i suoi confini vengono connessi al terreno, formando dei poligoni addizionali, i quali rappresentano le superfici laterali del solido, o le pareti dei confini interni.

Questo metodo è però funzionale solo se il solido ha un'unica superficie superiore; in caso contrario, il solido deve essere scomposto in componenti con una sola superficie superiore. Per descrivere invece il solido attraverso l'insieme delle superfici che ne compongono il **boundary** è necessario raggruppare le superfici all'interno di una geometria composta (*MultiGeometry*), disabilitando l'estrusione. Il sistema descritto rende più controllabile la generazione delle facce del solido, ma, d'altra parte, richiede un lavoro manuale di scomposizione delle entità tridimensionali.

3.2.7 COLLADA

COLLADA (COLLAborattive Design Activity) è un formato file di interscambio tra applicazioni 3D distribuito gratuitamente insieme al codice sorgente dall'organizzazione no-profit Kronos Group Inc. Il formato file è realizzato in codice XML. COLLADA definisce uno database schema XML che consente alle applicazioni di authoring in 3D di scambiare liberamente le risorse digitali senza perdita d'informazione. COLLADA non è semplicemente una tecnologia perché da sola non riesce a risolvere i problemi di comunicazione. COLLADA è riuscita a fornire una zona neutrale in cui i concorrenti possono lavorare insieme nella progettazione di una specifica comune.

Il COLLADA schema supporta tutte le feature che le moderne applicazioni interattive 3D necessitano, compresi gli effetti *shader* programmabili e la simulazione fisica. Può essere anche facilmente estesa da parte degli utenti finali per i loro specifici scopi. In altre parole, COLLADA non è progettato per essere un temporaneo meccanismo di trasporto dei dati, ma piuttosto quello di essere lo schema per i dati di origine per le risorse digitali. Non è concepito come un meccanismo di distribuzione, ma per essere un

supporto ai contenuti per qualsiasi piattaforma di destinazione. La scelta di COLLADA ad utilizzare il linguaggio XML, ha fatto in modo di poter conquistare così molti dei benefici della eXtensible Markup Language incluso il supporto nativo di codifica internazionale UTF-8. Il documento XML Schema di COLLADA è pubblicamente accessibile su Internet per la validazione dei contenuti online.
di seguito le principali caratteristiche:

- importazione di geometrie mesh;
- gerarchie di trasformazione;
- Materiali e texture;
- Shader;
- Skin e Morphing;
- Animazione;
- Simulazioni fisiche;
- istanze;

Inserire modelli COLLADA

Dalla versionee KML 2.1 è possibile importare modelli 3D come edifici, ponti, monumenti, e statue realizzate con il formato di interscambio file COLLADA. I modelli sono definiti in modo indipendente di Google Earth nel proprio spazio di coordinate, utilizzando applicazioni come SketchUp, 3D Studio Max, Softimage XSI, o Maya. Nello specifico l'elemento <model> dà la possibilità di inserire un oggetto 3D definito in un file COLLADA (referenziato nel tag <link>). I file COLLADA usano come estensione l'estensione '.dae'. I modelli sono creati in un proprio spazio di coordinate e poi localizzato, posizionato, e scalato in Google Earth. Google Earth supporta i profili COLLADA comuni, con le seguenti eccezioni:

- supporta solo triangoli e linee come tipi primitivi. Al massimo il numero di triangoli permessi e 21845.
- Google Earth non supporta animazioni o *skinning*.
- Google Earth non supporta riferimenti a geometrie esterne.

Di seguito alcuni elementi specifici di Model:

<altitudeMode> specifica come <altitude> specificata in <Location> è interpretata possibili valori

- **clampToGround** - (default) indica di ignorare la <altitude> specificata e piazza il Model sopra la superficie dell terra.
- **relativeToGround** - Interpreta <altitude> come un valore in metri sopra la superficie della terra.
- **absolute** - Interpreta <altitude> come un valore in metri sopra il livello del mare.

<Location> Specifica le coordinate esatte di origine del modello espresse in latitudine e longitudine e altitudine. Le misure di latitudine e longitudine sono proiezioni standard Lat-Ion con datum WGS84. Altitudine è una distanza rispetto la superficie della terra, in metri, e in accordo con l'interpretazione data da <altitudemode>.

<Orientation> Descrive la rotazione del sistema di coordinate ddel modello 3D per posizionare l'oggetto in Google Earth. Le rotazioni possibili sono specificate dai tag <heading>, <tilt> e <roll> che apportano rotazioni come mostrato nella Figura 3.31.

<scale> Scala le coordinate del modello lungo gli assi x,y e z.

<link> specifica il link dove caricare il file del modello

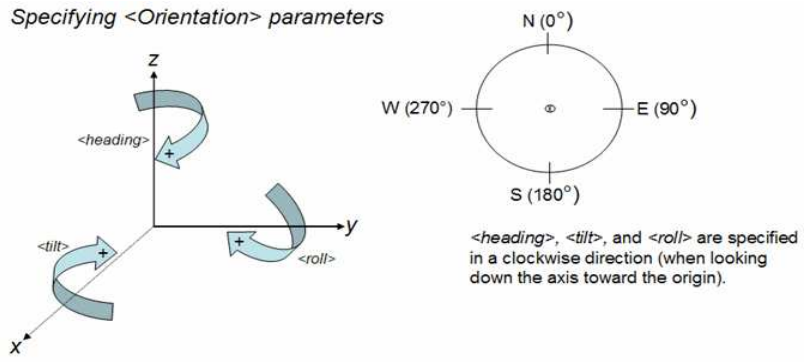

Figura 3.31: parametri per l'orientamento del modello

3.2.8 Stili

Google Earth e gli altri Earth browser mettono infine a disposizione strumenti per la personalizzazione dell'informazione geografica, come gli stili per il testo, i colori e le icone, utilizzabili per l'individuazione dei luoghi all'interno di segnalibri geografici. L'elemento astratto StyleSelector, è la tipologia base per gli elementi <Style> e <StyleMap>. L'elemento StyleMap sceglie uno stile basato sulla modalità corrente del Placemark. Un elemento derivato da StyleSelector è univocamente identificato dal suo id e dal relativo URL.

Style

Uno elemento <Style> definisce un gruppo indirizzabile di stili che possono essere referenziati attraverso <StyleMap> o da una Feature. Gli stili hanno effetto su come una geometria viene rappresentata in un visualizzatore 3D e come le Feature appaiono nella lista del pannello Luoghi in Googlee Earth. Per poter condividere una collezione di stili in un <Document> è necessario che ad ognuno di loro sia definito un ID così che possano essere referenziati dalle singole Feature che gli utilizzano. Le Feture come <Placemark> per poter referenziarsi allo <style> devo utilizzare <styleUrl> il cui valore sarà l'ID dello stile voluto.
Principali elementi derivati da <style>:

BalloonStyle Specifica come la descrizione del Balloon per i placemark è disegnata. Il balloon è un fumetto che si apre quando si clicca sopra ad un oggetto in Google Earth.

<IconStyle> Specifica come le icone per un punto placemark vengono disegnate nel pannello luoghi di Google Earth.

<LabelStyle> Specifica come il nome di una Feature inserito nell'elemento <name> è disegnato in Google Earth.

<LineStyle> Specifica lo stile (colore, spessore) per tutte le linee appartenenti ad una geometria.

<ListStyle> Specifica la modalità con la quale le Feature vengono visualizzate nella lista. La lista è una gerarchia container e figli; in Google Earth, questa lista e il pannello Luoghi.

<PolyStyle> specifica lo stile con il quale si disegnano i poligoni.

StyleMap

Un elemento <StyleMap> mappa tra due stili diversi. Tipicamente è usato per fornire separati stili per un placemark nel caso normale e nel caso highlighted, la versione highlighted appare quando l'utente si posa sopra all'oggetto con il mouse in Google Earth.
Gli elementi specifici di StyleMap sono:

Pair elemento obbligatorio definisce una coppia chiave/valore che mappa un modo (normale / highlighted) per il predefinito stile.

- **<Key>** che identifica il modo
- **<styleUrl> or <Style>** che identifica lo stile.

Di seguito un esempio di utilizzo degli elementi <Style> e <MapStyle>:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document>
  <name>StyleMap.kml</name>
  <open>1</open>
  <Style id="normalState">
    <IconStyle>
      <scale>1.0</scale>
      <Icon>
        <href>http://maps.google.com/mapfiles/kml/pal3/icon55.
            png</href>
      </Icon>
    </IconStyle>
    <LabelStyle>
      <scale>1.0</scale>
    </LabelStyle>
  </Style>
  <Style id="highlightState">
    <IconStyle>
      <Icon>
        <href>http://maps.google.com/mapfiles/kml/pal3/icon60.
            png</href>
      </Icon>
      <scale>1.1</scale>
    </IconStyle>
    <LabelStyle>
      <scale>1.1</scale>
      <color>ff0000c0</color>
    </LabelStyle>
  </Style>
  <StyleMap id="styleMapExample">
    <Pair>
      <key>normal</key>
      <styleUrl>#normalState</styleUrl>
    </Pair>
    <Pair>
      <key>highlight</key>
      <styleUrl>#highlightState</styleUrl>
    </Pair>
  </StyleMap>
  <Placemark>
    <name>StyleMap example</name>
    <styleUrl>#styleMapExample</styleUrl>
    <Point>
      <coordinates>-122.368987,37.817634,0</coordinates>
    </Point>
  </Placemark>
</Document>
</kml>
```

Capitolo 4

Stato dell'arte e relazione con altri progetti

4.1 3D GeoDatabase per la città di Berlino

Nel periodo da Novembre 2003 a Dicembre 2005 viene commissionato ufficialmente dal Senato di Berlino e dalla Berlin Patner GmbH il progetto per la realizzazione di un modello virtuale per la città di Berlino, un progetto finanziato dall'Unione Europea. Il modello gioca un ruolo importante nell'infrastruttura di dati spaziali tridimensionali di Berlino e apre a una moltitudine di applicazioni per il settore pubblico e privato.

Il progetto permette l'upgrade del modello virtuale per la città di Berlino ufficiale basandolo sulle più recenti specifiche CityGML 0.4.0 nel 2007. Maggiori estensioni al modello sono realizzate modellandolo all'interno (aggiunto livelo di dettaglio Lod-4) e la visualizzazione delle informazioni di superficie variabili nel tempo in accordo con il modello appearance di CityGML. Nell'agosto 2008, CityGML 1.0.0 diventa uno standard adottato da consorzio OGC.

4.1.1 Progetto 3D City DB

Dall'esperienza sopra citata nasce il '3D Geo Database basato su CityGML' ora rinominato '3D City DB' sviluppato da Thomas H. Kolbe, Gerhard König, Claus Nagel, Alexandra Stadler presso l'istituto *'Institute for Geodesy and Geoinformation Science, Technische Universität Berlin'*.

Il '3D Geo Database basato su CityGML' permette la memorizzazione, la gestione e la manipolazione di modelli 3D di città con l'uso di un database spaziale, inoltre mette a disposizione un tool per l'importazione e l'esportazione di dati raster e orthophoto. Il 3D Geo Database è stato realizzato con un schema per un data base relazionale Spaziale Oracle 10G R2. In una prima fase del progetto, hanno costruito una prima versione che era limitata ad un sottoinsiemi di CityGML. In una seconda fase, è stato ridisegnato lo schema del Database per rispettare pianamente CityGML 1.0.0 . In dettaglio il database implementa le seguenti feature chiave di CityGML:

- **modellazione Complexthematic** la descrizione tematica include attributi, relazioni e aggregazioni gerarchiche annidate tra oggetti. Il gran numero di informazioni tematiche può essere usato per interrogazioni tematiche, analisi o simulazioni.
- **Cinque livelli di dettaglio (LoD)** ogni oggetto geografico può essere memorizzato in cinque diversi livelli di dettaglio. All'aumentare del LoD, gli oggetti non solo acquisiscono una geometria più fine e precisa, ma guadagnano precisione anche a livello tematico.
- **Dati Appearance** oltre a informazioni semantiche e geometriche, gli oggetti possono avere anche appearance, ovvero proprietà osservabili della superficie dell'oggetto.
- **DTM Digital Terrain Model** il DTM nel geo database può essere rappresentato in quattro modi diversi: reticolo regolare, TIN (Triangular Irregular Network), 3D mass point e 3D break line. Per ogni LoD un complesso relief(rilievo)può essere aggregato da un numero qualsiasi di componenti DTM e di differente tipo.
- **Rappresentazione di oggetti generici e prototipi** i prototipi sono usati per avere una maggiore efficienza nella memorizzazione e nella manipolazione degli oggetti, che appaiono frequentemente nel modello. Ogni istanza di un prototipo può riferirsi ad un oggetto per ogni LOD.
- **Aggregazione anche gerarchica di oggetti geografici** gli oggetti geografici possono essere aggregati in gruppi in accordo a criteri stabiliti dall'utente, ad esempio per modellare un complesso di edifici composto da un insieme di singoli edifici. Ogni gruppo può contenere altri gruppi, risultando in una aggregazione di profondità arbitraria.
- **Geometrie 3D** le geometrie degli oggetti 3D possono essere rappresentate attraverso una combinazione di superficie e solidi così come una aggregazione , anche ricorsiva, di tutti questi elementi.

Il Database prevede anche due funzionalità aggiuntive rispetto a CityGML, queste funzionalità sono:

- **Gestione orthophoto** usando le funzionalità di Oracle 10g R2 Spatial GeoRaster il database è in grado di trattare orthophoto.
- **Gestione della versione e della cronologia** usa il Workspace Manager di Oracle.

4.1.2 Tool di Importazione/Esportazione

Il tool 3D City DB Importer/Exporter è il front-end per il 3D City Database e ha come principale funzione quella di importare ed esportare dati in formato CityGML. Le principali caratteristiche che sviluppa sono le seguenti:

- Pieno supporto all versioni CityGML 1.0.0 e 0.4.0
- Lettura/scrittura di istanze di documenti CityGML con arbitraria grandezza dei file.
- programma Multithread facilitando così le alte performance di processo.
- Supporto per CityGML appearrance come texture e matirial
- Risoluzione in avanti e in indietro degli XLink
- Matching/mearging delle feature degli edifici all'interno del database.

E' completamente sviluppato in linguaggio JAVA, con l'uso della tecnologia JDBC per comunicare con il database e dalla libreria citygml4j per mappare gli oggetti CityGML in oggetti JAVA e viceversa. Anche la libreria citygml4j è sviluppata dallo stesso gruppo che ha implementato sia 3D City DB che il 3D City DB Importer/Exporter, questa libreria rende più facile la leggere, processare, e scrivere dataset CityGML, e lo sviluppo di applicazioni software basate su CityGML.

Il tool, oltre alle operazioni di importazione e esportazione dei modelli CityGML e dei dati raster alle quali possono essere applicati, ha anche la possibilità di fondere dei modelli CityGML con la funzione matching. Questa funzione consente di memorizzare due modelli diversi della stessa città nello stesso database, fondendoli in uno solo, cosi ottenendo un di avere un modello più aggiornato e più completo. Ad esempio si può fondere due modelli ad esempio il primo con un livello di dettaglio inferiore rispetto al secondo. I due diversi dataset possono essere distinti attraverso l'attributo *lineage* specificato prima di effettuare l'importazione del modello. Il processo di matching si basa su due passi:

1. **Confronto della posizione degli edifici** ogni edificio 3D è appiattito in uno 2D, omettendo la sua altezza. I due edifici da fondere sono comparati in base della loro sovrapposizione.
2. **trovare edifici corrispondenti** un percentuale minima per entrambe le direzioni è usata per determinare le coppie di edifici corrispondenti.

Nella versione in sviluppo è stata aggiunta anche la possibilità di esportare i modelli contenuti all'interno del databse 3D City DB in formato KML dando la possibilità di scegliere quattro modalità di visualizzazione:

- **Footprint** gli edifici sono rappresentati attraverso la loro proiezione sulla superficie terrestre risultando cosi appiatti al suolo
- **Extruded** gli edifici sono rappresentati come un modello a blocchi usando la tecnica dell'estrusione della rappresentazione footprint attraverso la conoscenza dell'altezza dell'edificio (inserita nell'attributo thematic di CityGML)

- **Geometry** mostra dettagliatamente la geometria del suolo, la superficie delle pareti e dei tetti degli edifici ai quali aggiunge delle informazione attraverso delle appearance. Ad esempio i tetti possono essere colorati di rosso e le pareti colorate di grigio. (KML nativamente non supporta le texture)

- **COLLADA** Per superare il problema di non poter utilizzare le texture KML può inserire dei modelli esterni realizzati con COLLADA che a sua volta permette l'utilizzo delle texture. Quindi questa visualizzazione permettete lo stesso dettaglio di geometry con in più la possibilità di supportare le texture.

è possibile associare ad ogni elemento placemark alcune informazioni come ad esempio l'indirizzo l'altezza attraverso l'uso dei Balloon (fumetti che appaiono se si clicca sopra il placemark) tipici dell'ambiente Google Earth.

Sia in importazione che nella fase di esportazione è data la possibilità di filtrare i modelli attraverso dei filtri che sono i seguenti:

- **GML_ID:** vengono importati e esportati solo gli oggetti che hanno lo stesso id indicato nel filtro.

- **BoundingBox:** vengono importati e esportati solo gli oggetti che sono racchiusi nell'area delimitata dal boundig box.

- **Feature:** vengono importate e esportate solo le feature selezionate.

Fasi principali dello sviluppo del progetto

Nella seguente sezione si evidenziano i differenti passi che hanno portato allo sviluppo del progetto 3D City DB i tre principali step sono portati in evidenza nella Figura 4.1

a) Semplificazione del modello di dati CityGML Al fine di conseguire uno schema per il database più compatto e migliorare le performance delle interrogazioni, il complesso modello di dati viene semplificato in alcuni punti critici.

b) Derivazione dello schema relazionale per il database La semplificazione del modello orientato agli oggetti, è mappato sulle tabelle dello schema relazionali. Il numero di tabelle è ottimizzato in modo da minimizzare il numero di join presenti tipicamente nelle interrogazioni.

c) Creazione di un tool per l'importazione l'esportazione il tool per l'amministrazione del database permette di processare istanze di documenti molto gradi (» 4 GB). Sistema multiprocessore o multi-core CPU sono sfruttata attraverso una architettura multithreaded.

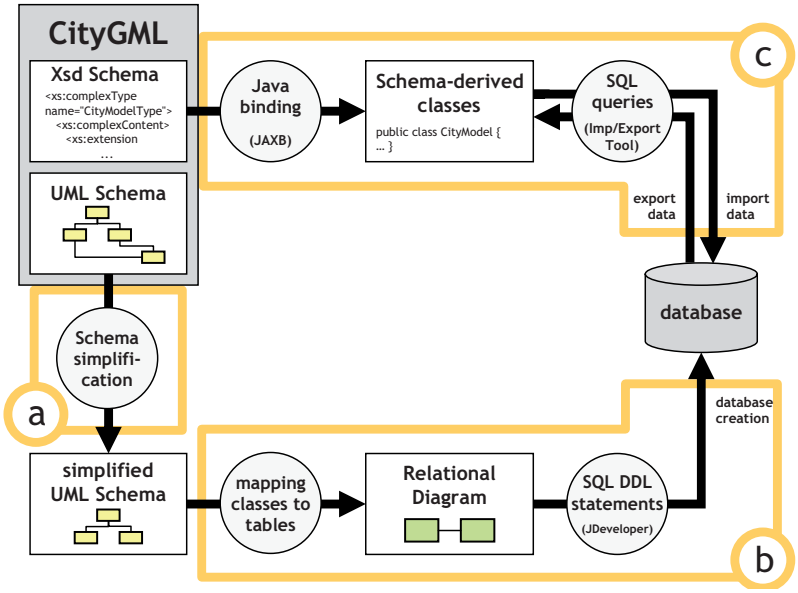

Figura 4.1: (a) Semplificazione del modello di dati CityGML, (b) Derivazione dello schema relazionale, (c) Creazione del Tool di Import/Export per le istanze dei documenti cityGML

4.2 Tool per l'importazione e l'esportazione di dati CityGML in PostGIS

Nel 2009 presso la facoltà di Ingegneria dell'università degli studi di Padova è stato sviluppato un tool per l'importazione, l'esportazione e la manipolazione di dati in formato standard CityGML basato sul progetto "3D City DB" per poter, a differenza del progetto sviluppato dall'università di Berlino, usare come DBMS spaziale non Oracle Spatial ma utilizzare un DBMS opensource come PostgreSQL integrandolo con l'estensione spaziale PostGIS che rende PostgreSQL un geodatabase al pari di Oracle Spatial [1]. I punti fondamentali sui quali questo progetto è intervenuto sono stati:

- reingegnerizzazione dello schema relazionale del database 3D City DB per adattare i tipi di dati di Oracle con quelli di PostgreSQL/PostGIS;
- Modifica del tool 3D City DB Import/Export;
  - realizzare la comunicazione del tool e database server con PostgreSQL/-PostGIS anziché con Oracle.
  - gestione della memorizzazione ed estrazione dei dati geometrici utilizzando i driver e le librerie specifiche per PostgreSQL/PostGIS in sostituzione a quelli utilizzati per Oracle;

- gestione della memorizzazione ed estrazione delle texture che, come la parte geometrica, presenta delle differenze tra Oracle e PostreSQL/Postgis.

Il progetto realizzato però non è riuscito a raggiungere lo stesso livello di prestazioni e completezza del progetto "3D City DB". Infatti implementa solo una gestione parziale delle funzionalità richieste. Le lacune principali sono le seguenti:

- non tutte le feature possono essere importate ed esportate in maniera corretta;
- si può importare ed esportare solo per i primi tre livelli di dettaglio;
- la gestione dei filtri in fase di importazione e esportazione non è presente;
- non implementata la gestione degli indici per poter avere in fase di esportazione una maggiore efficienza;
- una interfaccia grafica non completa;

Inoltre il progetto è rimasto statico e non ha seguito gli aggiornamenti che invece il progetto 3D City DB seguito *'Institute for Geodesy and Geoinformation Science, Technische Universität Berlin'* ha portato avanti in questi ultimi anni.

4.3 Sviluppo e progettazione del Toolkit

La progettazione e lo sviluppo del toolkit in esame in questa tesi poggia le proprie fondamenta sulla progetto *3D City DB* e parte da quanto già realizzato a dall'Università Degli Studi di Padova [1] che viene completamente rivisto per aggiornare, completare ed estendere schema dati e funzionalità.

In una prima fase si è dovuto intervenire sui processi di importazione ed esportazione per andare a completare il porting da Oracle verso PostgreSQL/PostGIS e rendere completa l'importazione e l'esportazione di tutte le feature a tutti e cinque i livelli di dettaglio. Sono stati aggiunti i filtri sia in fase di importazione e sia in fase di esportazione. É stato necessario modificare sia le procedure di memorizzazione geometrica sia lo schema del database relazionale riprendendo in parte la struttura dello schema realizzato nel progetto *3D City DB*. Si è ripresa l'interfaccia grafica ed è stata rielaborata integrandola con quella realizzata nel tool *3D CityDB Import/Export*. Il database è stato, dove necessario, indicizzato sia con indici geometrici sia con indici non geometrici.

In una seconda fase si è passati ad implementare tutte quelle funzionalità che nella versione più recenti di 3D City DB sono state aggiunte. Le funzionalità aggiunte sono le seguenti:

- possibilità di inserire le coordinate del bounding box con sistema di riferimento diverso da quello utilizzato per la memorizzazione delle geometrie nel database

- possibilità in fase di esportazione di dividere attraverso la funzionalità tile bounding box, che seziona in righe e colonne definite dall'utente, l'intera area da esportare in un insieme di sottoaree. Per ogni sottoarea corrisponde un file CityGML che rappresenta la stessa.
- possibilità di calcolare il massimo bounding box di tutte gli oggetti appartenerti al database o ad una specifica categoria di feauture che l'utente può selezionare.
- poter scegliere in fase di esportazione di estrarre i dati in un sistema di riferimento diverso da quello utilizzato nel database;
- esportare i modelli contenuti nel database non solo nel formato CityGML ma anche in formato KML.

Capitolo 5

Tecnologie utilizzate

5.1 PostgreSQL con estensione PostGIS

PostgreSQL è un sistema di gestione di database relazionale ad oggetti (ORDBMS) basato su POSTGRES, Versione 4.2, sviluppato dalla "University of California", nel dipartimento di informatica Berkeley. POSTGRES proponeva molti concetti che diventarono disponibili solo in alcuni sistemi di database commerciali molto più tardi. PostgreSQL è il discendente opensource di quel codice originale Berkeley. Supporta una parte molto grande dello standard SQL e offre molte altre funzionalità:

- query complesse
- chiavi esterne
- trigger
- viste
- integrità transazionale
- controllo concorrente multiversione

Inoltre, PostgreSQL può essere esteso dall'utente in molti modi, per esempio aggiungendo nuovi tipi di dato, funzioni, operatori, funzioni aggregate, metodi di indice, linguaggi procedurali. Data la licenza libera, PostgreSQL può essere usato, modificato e distribuito da chiunque gratuitamente per qualsiasi scopo, sia esso privato, commerciale, o accademico. Per molti anni lo sviluppo di PostgreSQL è stato possibile solo in ambienti UNIXlike. A partire dalla versione 8.0 gira in maniera nativa anche in ambienti Microsft Windows.

L'estensione spaziale PostGIS nasce nel 2001 nell'azienda canadese Refrections di Victoria, British Columbia, specializzata nello sviluppo GIS. Che per le loro esigenze di fare uso di DBMS con estensione geografiche decise di sviluppare un'estensione spaziale per PostgreSQL. PostGIS ricopre il 100% delle funzionalità SQL Spatial indicate nelle specifiche "Simple Feature SQL" rilasciate dall'OGC. Queste specifiche oltre

a definire la sintassi SQL per i calcoli sulle relazioni spaziali fra gli oggetti geometrici, introdicone delle strutture dati relative all'archiviazione di una singola geometria. Queste strutture sono rappresentati da due formati:

WKT (WellKnown Text) rappresentazione testuale creato per rappresentare una geometria vettoriale su una mappa, un sistema di Coordinate di Riferimento o una trasformazione tra sistemi di coordinate.

WKB (WellKnown Binary) ha la stessa rappresentazione di WKT; la differenza è che la sintassi, invece di essere basata sul testo, e espressa con valori di byte.

Le specifiche "Simple Feature SQL" permettono la memorizzazione nel database di sette tipi diversi di dati geometrici:

1. POINT
2. LINESTRING
3. POLYGON
4. MULTIPOINT
5. MULTILINESTRING
6. MULTIPOLYGON
7. GEOMETRYCOLLECTION

Ogni tipo a la propria sintassi sia su piani bidimensionale che tridimensionali. Ad ogni geometria deve essere indicato un proprio sistema di riferimento SRID (Spatial Reference System Identifier). SRID deve essere scelto tra quelli presenti nella tabella di servizio generata da PostGIS *spatial_ref_sys* nella quale sono specificati tutti gli SRID utilizzabili. Nel caso di SRID ignoto si usa il valore 1.

Per poter ottimizzare le prestazioni dell'accesso ai dati, è possibile associare alla colonna che contiene i dati geometrici un indice spaziale. PostGIS implementa indici spaziali del tipo GIST (Generalized Search Tree).

L'inserimento dei dati geometrici può avvenire in due modalità:

- creazione comandi SQL secondo la sintassi *Simple Features SQL*,
- importazione dei dati attraverso tool testuali o grafici.

Nel caso si utilizzi la prima tipologia si crea prima la tabella secondo la classica sintassi e in secondo luogo aggiungere una colonna contenete i dati geometrici di seguito un esempio:

```
CREATE TABLE punti(ID int4,Descrizione varchar(40));
SELECT AddGeometryColumn('punti','punto', 26591, 'POINT', 2);
```

Dove, i parametri della funzione AddGeometryColumn sono

- 'punti' è il nome della tabelle a cui aggiungere la colonna dei dati geografici,
- 'punto' il nome della colonna delle geometrie,
- 26591 è lo SRID,
- 'POINT' è il tipo di geometria
- 2 la dimensione che indica una geometria bidimensionale, se fosse stata tridimensionale avremmo inserito 3

L'inserimento delle singole geometrie avviene poi, secondo le classiche istruzioni SQL di INSERT in cui, l'istruzione legata alla geometria segue lo schema WKT. Ad esempio:

```
INSERT INTO 'punti' ('punto', descrizione)
VALUES (POINT(0 0 0), 'Punto␣inserito');
```

PostGIS mette a disposizione tutta un serie di funzioni che ci aiutano a manipolare i dati geometrici che sono classificate nelle seguenti categorie:

Funzioni di base Permettono la creazione/eliminazione di colonne geometriche e l'attribuzione dei dati ad un determinato sistema di riferimento;

Funzioni di relazioni fra geometrie Permettono di calcolare area, perimetro, buffer, centroide, ecc., di una data geometria. Permettono inoltre di effettuare operazioni di unione e sottrazione tra geometrie;

Funzione di "informazioni" sulle geometrie Attraverso di esse è possibile conoscere il tipo di geometria presente in un dato campo, l'ID del sistema di riferimento utilizzato, le coordinate dell'ultimo punto di una geometria, il numero di punti contenuti, il valore x o y o z di un dato record, ecc. Permettono inoltre di visualizzare i dati geografici in formato WKB quando siano archiviati in WKT, e viceversa (rendendo, ad esempio, intellegibili i dati geografici archiviati in formato binario);

Funzioni di creazioni di geometrie Permettono di creare dati geometrici e di archiviarli in formato standard OGC a partire da un insieme di coordinate. Questo permette di trasformare in un database geografico un database alfanumerico in cui le coordinate dei punti siano, ad esempio, archiviate in due campi 'x' e 'y';

Oltre alle funzioni sopra descritte, e che fanno parte di quelle standard OpenGIS, PostGIS ha numerose funzioni avanzate. Ne riassumiamo alcune tra le più importanti:

Funzioni di calcolo di misure Permettono di calcolare le aree, i perimetri, le lunghezze, gli sferoidi, ecc., in relazione al sistema di riferimento e al tipo di piano (bidimensionale o tridimensionale);

Funzioni di output Attraverso queste funzioni è possibile trasformare le geometrie al formato WKT al formato WKB (e viceversa), esportarle verso formati XML quali l'SVG (Scalable Vector Graphics; un XML per la creazione di geometrie su browser definito dal W3C) o il GML (un XML, definito dall'OGC, per l'archiviazione di dati GIS);

Funzioni di modifica delle geometrie Permettono di manipolare le geometrie effettuando ad esempio operazioni di semplificazione, di traslazione, di inversione, di conversione da un sistema di riferimento ad un altro, ...

Oltre a queste ci sono funzioni che forniscono informazioni strettamente legate a PostGIS; ad esempio permettendo di verificarne la versione, le opzioni (ad esempio se la versione in uso offre il supporto GEOS e/o PROJ), ... Numerosi operatori logici, infine, (di eguaglianza, sovrapposizione, ...) permettono di effettuare confronti sia tra le geometrie che tra i risultati di query anche complesse.

Le versione che si sono utilizzate per lo sviluppo del toolkit sono le seguenti:

- PostgreSQL versione 8.4
- PostGIS versione 1.5.2

5.2 Librerie JAVA

Il linguaggio di programmazione utilizzato per sviluppare il toolkit è il linguaggio JAVA. É stato scelto per le sue caratteristiche

- essere completamente orientato agli oggetti
- essere indipendente dalla piattaforma
- contenere strumenti e librerie per il networking
- essere progettato per eseguire codice da sorgenti remote in modo sicuro

inoltre sono disponibili tutta una serie di librerie e API progettate per questo ambiente che sono necessarie allo sviluppo del software in esame:

- **Driver JDBC per PostgreSQL/PostGIS** API e Driver per l'interfacciamento con il DBMS
- **JAXB** API per la lettura e scrittura e manipolazione di file in formato XML
- **citygml4j** API e libreria specifica per la lettura scrittura e di istanze CityGML.
- *GeoTools* toolkit GIS che fornisce l'implementazione di molte specifiche OGC

5.2.1 Driver JDBC per PostgreSQL/PostGIS

JDBC *Java DataBase Connectivity* è un connettore per database che consente l'accesso alle basi di dati da qualsiasi programma scritto con il linguaggio di programmazione Java, indipendentemente dal tipo di DBMS utilizzato. È costituita da un API, raccolta nel package java.sql, che serve ai client per connettersi a un database. Fornisce metodi per interrogare e modificare i dati. È orientata ai database relazionali ed è Object Oriented. La piattaforma Java 2 Standard Edition contiene le API JDBC, insieme all'implementazione di un bridge JDBC-ODBC, che permette di connettersi a database relazionali che supportino ODBC. Questo driver è in codice nativo e non in JAVA. JDBC è un'API a livello SQL, ossia consente l'utilizzo di comandi SQL occupandosi di tutti i dettagli per la loro traduzione, invio e esecuzione sul DBMS.

Può essere suddivisa in due parti fondamentali:

- Application API - Comprendono tutti gli elementi che un'applicazione Java usa per la connessione al database;
- Driver API - Forniscono la base per la scrittura di un driver specifico per un determinato motore relazionale (es. PostgreSQL).

Le Driver API riconoscono quattro modalità di implementazione differenti, chiamati livelli o tipo dei driver JDBC:

1. *Driver nativo*: questa soluzione prevede la realizzazione di funzioni JAVA che convertono le richieste JDBC in richieste ad un driver in codice nativo della macchina che esegue il programma Java. Il driver è specifico per il DBMS cui si vuole accedere ed è normalmente realizzato per permettere l'accesso al DBMS da parte di applicazioni scritte nei tradizionali linguaggi di alto livello.
2. *Bridge JDBC/ODBC*: questa architettura prevede di tradurre le richieste JDBC in richieste al driver manager ODBC. Questa soluzione richiede, quindi, che la macchina che esegue il codice JAVA possieda un'installazione di ODBC con il driver specifico per il DBMS cui si vuole accedere.
3. *Middleware-server*: soluzione che prevede l'uso di un server scritto in JAVA responsabile di tradurre le richieste provenienti dal driver manager nel formato riconosciuto dal particolare DBMS. Sul prodotto sono presenti diverse soluzioni che realizzano queste funzioni per interagire con i database relazionali più diffusi.
4. *Driver JAVA*: prevede l'uso di un driver specifico per il particolare sistema relazionale, in modo analogo ai driver in codice nativo usati in ODBC. I driver vengono normalmente offerti come opzioni dei produttori dei sistemi relazionali.

Le prime due soluzioni non sono realmente portabili, perché richiedono l'utilizzo di componenti nativi, cioè specifici dell'ambiente in cui vengono eseguiti. Al contrario, le ultime due soluzioni si appoggiano su un ambiente completamente JAVA.

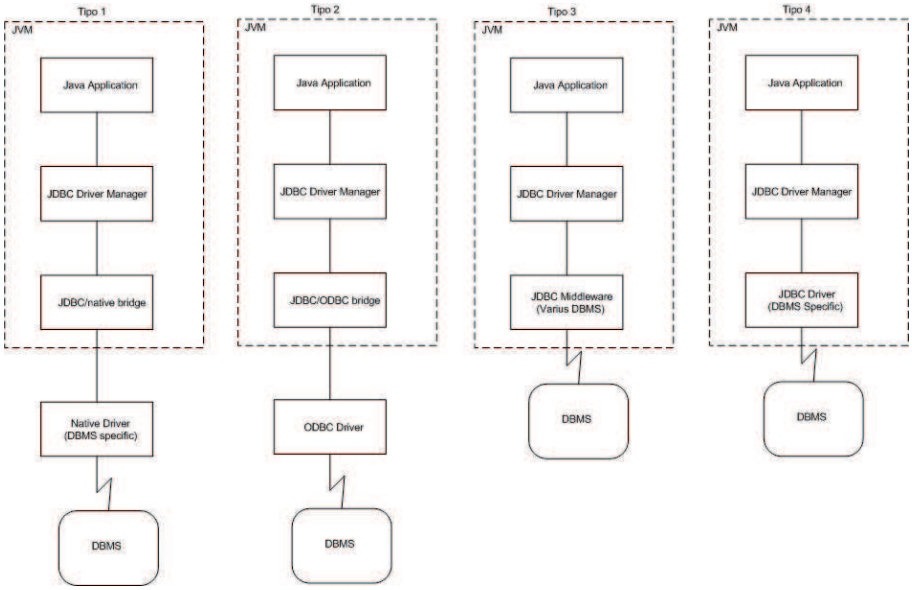

Figura 5.1: architettura driver jdbc

API e Driver JDBC per PostgreSQL/PostGIS PostgreSQL fornisce un driver JDBC conforme ai livelli 2, 3, 4, Nel progetto sviluppato si è usato il driver di tipo 4, ovvero quelli scritti interamente in JAVA, che sono supportati dalla JDK 1.6.0. In questo modo il driver è indipendente dalla piattaforma utilizzata.

L'estensione JDBC di PostGIS fornisce oggetti Java corrispondenti a tipi di dato interni di PostGIS come *Geometry, Box3D, Box2D* . Questi metodi possono essere utilizzati per scrivere client JAVA i quali interrogano il database PostGIS e disegnano o fanno calcoli su dati GIS in PostGIS. Per poter correttamente sviluppare è necessario caricare le librerie JDBC per PostgreSQL e per PostGIS. Nello sviluppo sono state usate le seguenti versioni:

- per PostgreSQL la versione 8.4-702
- per PostGIS la versione 1.5.0

5.2.2 JAXB

Java mette a disposizione diversi modi per accedere a file XML. Esistono infatti diverse API riguardanti parser SAX e DOM che permettono di leggere un file XML e analizzarlo in modo differente ottenendo il contenuto del file. Quello messo a disposizione da JAXB è un approccio alternativo al parsing. In SAX il parsing inizia dall'inizio del documento e passa all'applicazione ogni pezzo del documento in sequenza. In memoria non viene salvato nulla. L'applicazione può agire sui dati che gli arrivano dall'applicazione, ma non può effettuare nessuna operazione "in memoria", ad esempio effettuare degli aggiornamenti sui dati in memoria e restituire i dati aggiornati al

file XML. In DOM il parser crea un albero di oggetti che rappresentano il contenuto e l'organizzazione dei dati nel documento. In questo caso, l'albero esiste in memoria. L'applicazione può navigare attraverso l'albero per accedere ai dati di cui necessita, e se necessario, modificarli. JAXB (JAVA Architecture for XML Binding) permette agli sviluppatori JAVA di effettuare il mapping tra classi e una loro corrispondente presentazione XML. JAXB fornisce la possibilità di serializzare oggetti JAVA in XML attraverso l'operazione di *marshalling* e di effettuare l'operazione inversa *unmarshalling*, cioè permette di ottenere a partire da un file XML la corrispondente rappresentazione a oggetti JAVA. JAXB quindi permette di manipolare file XML senza la necessità di implementare alcuna routine specifica per il salvataggio e la lettura dei dati. Il pacchetto JAXB include il compilatore *xjc*, che viene usato per convertire XML Schema e altri formati di descrizione come XML DTD in classi JAVA. Il meccanismo di mapping tra XML schema e JAVA viene implementato mediante l'uso delle annotazioni definite nel package javax.xml.bind.annotation. Ad esempio @XmlRootElement e @XmlElement annotano le classi Java che rappresentano, rispettivamente, l'elemento radice del file XML e un elemento generico. n aggiunta a *xjc*, JAXB fornisce anche *schemagen*, un compilatore che effettua l'operazione inversa di *xjc*, cioè la generazione di un file XML Schema a partire da un insieme di classi Java annotate secondo le specifiche di JAXB. Vediamo in dettaglio come opera JAXB, come si vede dalla Figura 5.2 il procedimento per accedere al documento XML è il seguente:

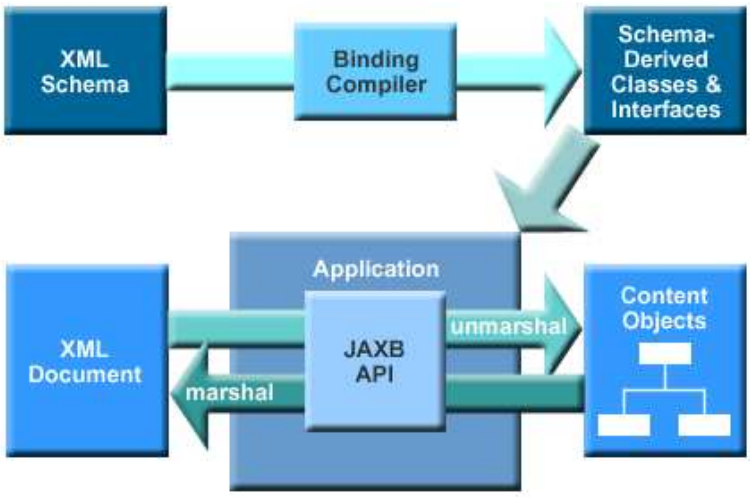

Figura 5.2: processo JAXB

- eseguire un *bind* dello schema del documento XML, ovvero genera le classi che rappresentano lo schema. attraverso il compilatore *xjc*.
- effettuare un *unmarshaller* del documento in oggetti JAVA, ovvero creare un albero degli oggetti contenuto all'interno del documento, diverso da quello creato

con DOM solitamente sono più efficienti. Questi oggetti sono istanze delle classi prodotte durante il binding dello schema.

Una volta eseguito *unmarshaller* del documento il programma può accedere ai dati contenuti nel file XML, semplicemente accedendo ai ai dati contenuti negli oggetti JAVA appena creati. Questo significa che lo sviluppatore può accedere ai dati XML senza averne una conoscenza specifica.

Per poter creare un file XML e scriverne il contenuto si devono effettuare queste operazioni:

- eseguire un *bind* dello schema del documento XML, ovvero genera le classi che rappresentano lo schema attraverso il compilatore *xjc*.
- creare un nuovo albero dei contenuti, che rappresenta il contenuto del nuovo file XML.
- eseguire un **marshaller** dell'albero nel file XML.

Una volta eseguita la funzione **marshaller** il contenuto dell'albero è scritto sul file XML. La versione utilizzata nel progetto è la 2.2.3

5.2.3 citygml4j

La libreria e API JAVA *citygml4j* permette di lavorare facilmente con i modelli CityGML. *citygml4j* rende più facile leggere, elaborare e scrivere set di dati CityGML e sviluppare applicazioni software basate su CityGML. *citygml4j* lega le definizioni dello schema XML di CityGML a un ben definito modello di oggetti JAVA Le istanze dei documenti CityGML sono deserializzate *unmarshall* in un corrisponde albero di oggetti JAVA che rappresenta il contenuto e l'organizzazione dell'istanza del documento. Questo approccio di *binding* consente agli sviluppatori JAVA di incorporare dati CityGML e funzioni di elaborazione in applicazioni JAVA senza dover conoscere XML o le sue API di elaborazione a basso livello. Invece di pensare all'analisi dei dati XML, gli sviluppatori possono concentrarsi sulla logica di business e di lavorare con oggetti CityGML ben definiti. Dopo l'elaborazione dei dati CityGML, l'albero dei contenuti JAVA può essere facilmente serializzati *marshall* in una istanza di un documento CityGML

citygml4j è open source e rilasciato sotto i termini della "GNU Lesser General Public License v3 (LGPL)" .

caratteristiche principali:

- Pieno supporto per la versione 1.0.0 e 0.4.0 CityGML.
- Supporto per il eXtensible Address Language (XAL).
- Il supporto per CityGML specifico sottoinsieme di GML 3.1.1
- Il supporto per la conversione tra diverse versioni CityGML.

- Integrazione con l'estensione per CityGML *Application Domain Exstensions* (ADE).
- Implementazione del *binding* sui dati XML basata su JAXB

La versione utilizzata è la 0.2.1 è non la più aggiornata in questo momento. Anche questa libreria fa parte del progetto 3D City DB seguito dall'IGGS dell'università di Berlino.

5.2.4 GeoTools

GeoTools è un open source JAVA toolkit GIS fornisce l'implementazione di molte specifiche OGC man mano che esse vengono sviluppate. GeoTools è anche associato con il progetto GeoAPI che crea interfacce JAVA geospaziali. Il codice di GeoTools è costruito utilizzando i più moderni strumenti e ambienti JAVA e la sua architettura modulare permette di aggiungere facilmente funzionalità aggiuntive. Il codice GeoTools è rilasciato sotto la licenza GNU Lesser General Public License (LGPL).

Principali caratteristiche:

- supporta lo standard OGC Grid Coverange Implementation
- Sistema di coordinate di riferimento e supporto alle trasformazioni
- simbologia utilizzando lo standard OGC Syled Layer Descrption SLD
- Supporto alle Java Topology Suite (JTS).
- attributi e filtri utilizzando le specifiche dei filtri OGC.
- supporta grafici e reti.

nel progetto viene utilizzato per poter fare una proiezione delle coordinate da un sistema di riferimento ad un'altro. Questa funzionalità è necessaria visto che le funzioni di trasformazioni rese disponibili da PostGIS sono realizzate per garantire le sole geometrie 2D e solo in parte per le geometrie 3D. Di seguito viene illustrato il codice utilizzato per la proiezione di geometrie espresse nel formato PostGIS utilizzando GeoTools. Il metodo ST_Trasform riproietta la geometria indicata nella variabile *geo* sul sistema di riferimento indicato nella variabile *SRID* espresso nel formato EPSG:

```
import org.geotools.referencing.CRS;
import org.opengis.referencing.crs.CoordinateReferenceSystem;
import org.opengis.referencing.operation.MathTransform;
import org.opengis.referencing.operation.TransformException;
import org.postgis.*;

public class Util_Transform {
  public static PGgeometry ST_Transform(PGgeometry geo, int
     SRID) throws Exception {
```

```
    Geometry geom = geo.getGeometry();
    if (geom == null) return null;
    CoordinateReferenceSystem sourceCRS = CRS.decode("EPSG:" +
       geom.getSrid(), true);
    CoordinateReferenceSystem targetCRS = CRS.decode("EPSG:" +
       SRID, true);
    MathTransform transform = CRS.findMathTransform(sourceCRS,
       targetCRS, false);

    for(int i=0; i<geom.numPoints(); i++){
        double src[];
        boolean hasZ = false;

        if (Double.isNaN(geom.getPoint(i).getZ()))
          src = new double[2];
        else {
          src = new double[3];
          hasZ = true;
        }

        src[0] = geom.getPoint(i).getX();
        src[1] = geom.getPoint(i).getY();

        if (hasZ)
          src[2] = geom.getPoint(i).getZ();

        try {
          transform.transform(src, 0, src, 0, 1);
        } catch (TransformException e) {}
        geom.getPoint(i).setX(src[0]);
        geom.getPoint(i).setY(src[1]);;

        if (hasZ)
          geom.getPoint(i).setZ(src[2]);;

    }

    // Todo: apply SRID to all the geometry subelements
    if (geom != null) {
      geom.setSrid(SRID);
    }
    geo.setGeometry(geom);
    return geo;
  }
}
```

É necessario precisare che comunque la riproiezione di coordinate 3D è un campo ancora in via di sviluppo ed è quindi possibile che anche l'utilizzo delle libreria Geoo-Tools possa in alcuni casi produrre dei risultati non corretti. La versione utilizzata è la

2.7

Capitolo 6

Disegno del modello dei dati

6.1 Semplificazione del modello CityGML

Per poter memorizzare i dati all'interno della base di dati è necessario creare uno schema relazionale del database che deriviamo dal diagramma UML delle classi di CityGML.

Da un'analisi condotta dal IGGS [2] per lo sviluppo del proprio tool è emerso che per l'elaborazione e la memorizzazione dei modelli nel database, è sufficiente uno schema semplificato. Queste semplificazioni sono apportate in modo tale da migliorare le performance in tempo di elaborazione dei dati. Andremo di seguito ad elencare queste semplificazioni.

Generalmente una o più classi dei diagrammi UML sono mappate in una tabella, dove il nome della tabella stessa corrisponde al nome della classe corrispondente. Gli attributi di una classe diventano le colonne della tabella corrispondente e il tipo di dato scelto e compatibile con quelli forniti da PostgreSQL. Nella Figura 6.1 si può vedere il passaggio dallo schema UML di CityGML allo schema relazionale dove in alcuno casi la trasformazione classe tabella non è sempre diretta.

In riferimento alle figure che rappresentano i vari modelli UML presentati nel paragrafo 3.1 per un intuitivo apprendimento le classi che saranno fuse in un'unica tabella sono contenute in un'area di colore arancio nel corrispondente diagramma UML. Le relazioni *N* a *m* le quali saranno rappresentate attraverso tabelle addizionali, sono contenute in aree di colore verde. Le parti blue invece sono parti che non sono completamente supportate da CityGML ma mappate lo stesso nel database.

Molteplicità, cardinalità e ricorsioni In cityGML gli attributi possono avere un numero variabile di occorrenze, questo viene tradotto nel database memorizzando i dati in un campo il quale permetta la memorizzazione di un valore arbitrario (ad esempio con una stringa che attraverso un separatore predefinito per separare le diverse occorrenze), o con array con un numero predefinito di oggetti che rappresentano il numero di oggetti coinvolti nell'associazione.

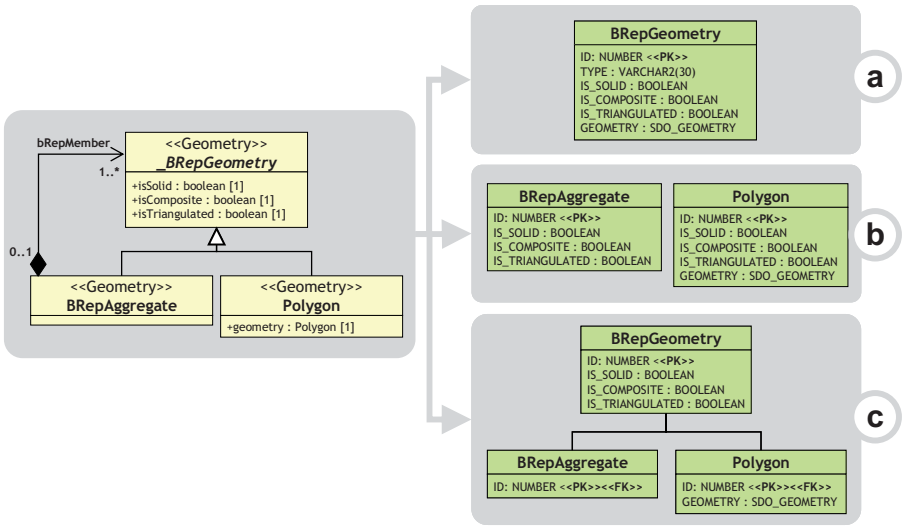

Figura 6.1: Mapping di una gerarchia di classi per le tabelle del database:(a) mappa tutte le classi in una singola tabella; (b) mappa le classi non astratte nelle proprie tabelle; mappa ogni classe in una singola tabella.

Le relazioni n:m presenti nel diagramma UML di CityGML, hanno necessità di essere sciolte con una tabella aggiuntiva. La tabella è costituita dalle chiavi primarie delle tabelle coinvolte nella relazione diventando assieme la chiave primaria della nuova tabella. Se la relazione 1:n e n:1, allora la tabella non serve.

Le associazioni ricorsive che hanno un costo elevato, per poter garantire buone prestazioni, sono implementati aggiungendo due colonne contenenti l'ID dell'elemento genitore(parent_ID) e quello dell'elemento radice (root_ID).

Tipo di dato I tipi di dato specifici di CityGML sono sostituiti da tipi di dati specifici a PostgreSQL. I dati di tipo geometry in CityGML sono sostituiti con tipo di dati geometrici di PostGIS come (POLYGON, LINESTRING, POINT,). String usato per rappresentare vettori di numeri e codici è stato sostituito con CHARACTER VARYRING così come le matrici. Le proprietà geometriche spaziale degli elementi sono rappresentate da oggetti geometrici di GML3 basati sul modello standard ISO 19107, in grado di rappresentare geometrie sia 2D che 3D. CityGML ne utilizza solo un sottoinsieme. Per le superficie 2D e 3D è usato un modello che memorizza tali geometrie come poligoni, che sono aggregati a MultiSurface, CompositeSurface,

Geometria La modellizzazione delle geometrie a due e tre dimensioni è modificata per eseguire alcune semplificazioni: Tutte le superficie basate sono memorizzate come poligoni, che sono aggregati in *MultiSurface, CompositeSurfaces, TriangulatedSurfaces, Solids, Multisolids* e infine *CompositeSolid*. Questa semplificazione permette di sostituire la rappresentazione più complessa usata per queste c lassi di GML3, con una

più semplice in modo tale da poter utilizzare una sola tabella SURFACE_GEOMETRY. Nel paragrafo 3.1.3 nella Figura 3.12 è rappresenta questa semplificazione.

6.2 Derivazione dello schema Relazionale per il geodatabase

6.2.1 Struttura schema relazionale

Modulo Core

Il core module contiene le tabelle che sono il cuore del database: cityobject, citymodel, external_reference, cityobjectgroup, generalization, surface_geometry. Di seguito riportiamo lo schema di questo modello nella Figura 6.3.

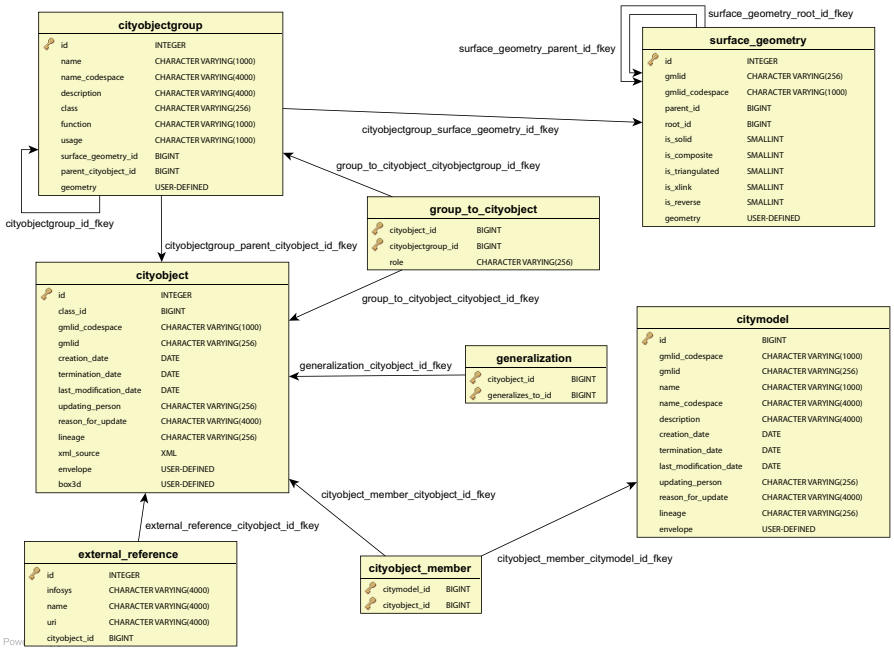

Figura 6.2: schema estratto dal database per il core model

vediamo in dettaglio le proprietà delle tabelle citate in precedenza:

- CITYOBJECT

  tutti i CityObject, e le istanze delle sue sottoclassi sono rappresentate da tuple in questa tabella. La proprietà Bunding Box viene rappresentata attraverso i due campi ENVELOPE e BOX3D, il primo rappresenta l'area di interesse specificata dal bounding box, rappresentandola con una geometria 2D, il secondo memorizza le coordinate in tre dimensioni del lower_left_corner e il upper_rigth_corner che individuano l'area di interesse del Bounding Box. La scelta di utilizzare questi campi è dovuta all'impossibilità da parte di PostGIS di trattare Envelope

in tre dimensioni, infatti la funzione ST_Envelope utilizzata crea solo inviluppi in due dimensioni tagliando completamente la terza dimensione se presente. Quindi per non perdere informazione è necessario utilizzare il campo BOX3D. Inoltre le operazioni topologiche come "Equals, Disjoint, Contains, Overlap,..." si possono effettuare solo per geometrie in due dimensioni in ambiente PostGIS. Quindi se vogliamo operare dei confronti tra inviluppi li possiamo fare solo in due dimensioni.

Per diminuire il numero di join durante le interrogazioni l'attributo GML_NAME non è presente, ma è stato aggiunto a tutte le sottoclassi. GMLID:CODESPACE è stato aggiunto e contiene l'indirizzo completo dell'oggetto tipicamente l'indirizzo del file importato. Gli attributi NAME e NAME_CODESPACE possono contenere più di un gml:name separati dalla stringa . CLASS_ID si comporta come sopra detto.

- CITYMODEL

  tutti i CityObject possono essere associati in un unico CityModel. Gli elementi appartenenti a questa classe sono usati come radice di una collezione CityGML. Ogni tupla sarà identificata da un ID auto incrementante.

- EXTERNAL_REFERENCE

  É usata per memorizzare i riferimenti esterni. La chiave esterna CITYOBJECT_ID fa riferimento ai CityObject associati. La sequenza EXTERNAL_REF_SEQ fornisce il valore ID disponibile per il prossimo EXTERNAL_REFERENCE.

- CITYOBJECTGROUP, GROUP_TO_CITYOBJECT

  Queste due tabelle realizzano il concetto di aggregazione. Il rapporto m:n tra un gruppo di oggetti (tabella CITYOBJECTGROUP), composto di oggetti contenuti in città CITYOBJECT è realizzato dalla tabella GROUP_TO_CITYOBJECT, che associa gli ID di entrambe le tabelle. La Tabella 6.1 mostra un esempio, in cui sono raggruppati due edifici di un complesso alberghiero.

La tabella principale nella quale memorizzare la rappresentazione geometrica degli oggetti CityGML è SURFACE_GEOMETRY. Anche in questo caso si discosta dalla rappresentazione UML offrendo comunque le stesse funzionalità. Ogni tupla della tabella SURFACE_GEOMETRY e memorizzato un poligono planare, con la possibilità di includere delle cavità. Ogni superficie può avere texture o un colore su entrambi i lati. Le texture sono memorizzate nelle tabelle che implementano appearance model. La base di una geometria 3D è il solido. L'ampiezza di un solido è definita da una superficie di contorno (finestra esterna). Una finestra è rappresentata da una superficie composta, in cui ogni finestra è usata per rappresentare un singolo componente del contorno del solido. Quest'ultimo consiste di superfici composte (una lista di superfici orientabili) connesse topologicamente in circolo. Le superfici possono essere messe in modo da formare un insieme di superfici o un perimetro di un oggetto volumetrico.

CITYOBJECTGROUP (estratto)

ID	NAME	NAME_ CODESPACE	DESCRIPTION	CLASS	FUNCTION	USAGE
1	Hotel complex	null	null	null	Building group	null

GROUP_TO_CITYOBJECT

CITYOBJECT_ID	CITYOBJECTGROUP_ID	ROLE
2	1	Main Building
4	1	Annex

CITYOBJECT (estratto)

ID	CLASS_ ID	GML_ ID	GML_ID_ CODE SPACE	LOWER_ LEFT_ CORNER	UPPER_ RIGHT_ CORNER	CREAT ION_ DATE	TERMINA TION_ DATE
2	26	Build1632	NULL	POINT()	POINT()	17.12.08	NULL
4	26	Build1633	NULL	POINT()	POINT()	17.12.08	NULL
1	23	Group	NULL	NULL	NULL	17.12.08	NULLL

Tabella 6.1: CityObjectGroup

L'aggregazione di più superfici, e.g. F_1 a F_n (ID dal 6 al 10 in Figura 6.3 e 6.4), è realizzata in modo tale che alle nuove superfici F_{n+1} (ID 2) non sia assegnata una geometria. Invece, il campo parent_id per le superfici da F_1 a F_n si riferisce all'ID di F_{n+1}.

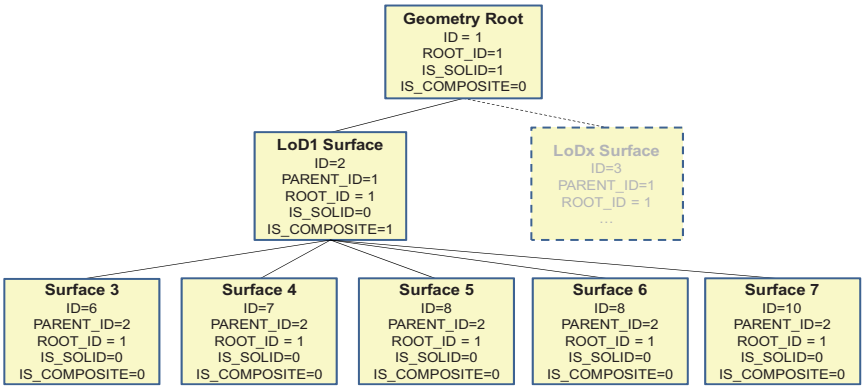

Figura 6.3: Geometria gerarchia si riferisce alla geometria solida in figura 6.4

Quindi viene aggiunta, una ulteriore tupla (ID 1) , la quale rappresenta il solido e definisce l'elemento radice dell'intera struttura aggregata. Ogni superficie è riferita alla sua radice, utilizzando l'attributo ROOT_ID. Queste informazioni hanno grande influenza sulle prestazioni del sistema, in quanto consente di evitare le query ricorsive. Se ad esempio vogliamo recuperare tutte le superfici, e gli elementi che formano un specifico edificio selezionato, basta semplicemente selezionare tutte quelle tuple che contengono la stessa ROOT_ID. Il lato negativo è la limitazione che ogni tupla in

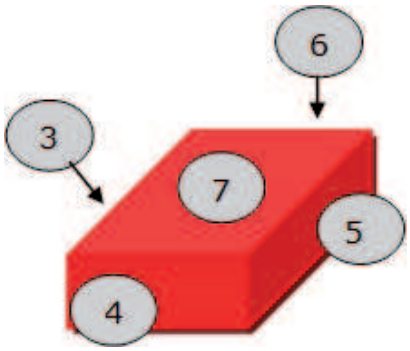

Figura 6.4: Edificio LoD1 volume chiuso composto da compositeSurface

SURFACE_GEOMETRY può appartenere solo ad un aggregazione.

Nella tabella SURFACE_GEOMETRY ci sono alcuni flag che caratterizzano il tipo di aggregazione come mostrato in Tabella 6.2: IS_TRIANGULATED indica una TriangulatedSurface, IS_SOLID distingue tra superficie (0) e solido (1), infine IS_COMPOSITE indica se si tratta di un aggregato (ad esempio: MultiSolid, MultiSurface) o di un composto (ad esempio: CompositeSolid, CompositeSurface).

	isSolid	**isComposite**	**isTriangulated**	**Geometry**
Polygon, Triangle, Rectangle				POLYGON()
MultiSurface				NULL
CompositeSurface		X		NULL
TraingulatedSurface			X	NULL
Solid	X			NULL
MultiSolid				NULL
CompositeSolid	X	X		NULL

Tabella 6.2: Attributi che determinano il tipo di aggregazione

Lo schema di aggregazione consente la definizione di aggregazioni nidificate (gerarchia delle componenti). Ad esempio, una geometria dell'edificio (CompositeSolid) può essere composto dalla geometria casa (CompositeSolid) e della geometria garage (Solid), mentre la geometria della casa è ulteriormente decomposto nella geometria del tetto (solido) e la geometria del corpo casa (Solid). Per fornire un identificatore univoco nella tabella SURFACE_GEOMETRY, il successivo valore disponibile come ID è previsto dalla sequenza SURFACE_GEOMETRY_SEQ.

In aggiunta sono inclusi in SURFACE_GEOMETRY due attributi: IS_XLINK e IS_REVERSE. Vediamo di seguito qual è il loro compito:

CityGML permette per condividere oggetti geometrici tra differenti geometrie o differenti caratteristiche tematiche usando XLINK concetto di GML3. Per questo motivo, agli oggetti condivisi è assegnato un *gml:id* univoco che può essere referenziato da una geometria GML attraverso l'attributo *xlink:href*. Il concetto permette di evitare la

ridondanza. Inoltre CityGML non utilizza il pacchetto di topologia costruito in GML3 ma usa piuttosto il concetto XLink per la modellazione esplicita della topologia.

Sebbene un XLink può essere visto come un puntatore ad un oggetto geometrico esistente, la tabella SURFACE_GEOMETRY non offre un attributo con chiave esterna che potrebbe essere usato per riferirsi a un'altra tupla all'interno della tabella. Il motivo di questo è che la tupla referenziata tipicamente appartiene ad una differente geometria aggregata. Pertanto, le chiavi esterne violerebbero il meccanismo di aggregazione della tabella SURFACE_GEOMETRY. Il modo migliore per risolvere gli XLink riferiti ad oggetti geometrici richiede due passi:

1. se la tupla riferita della tabella SURFACE_GEOMETRY deve essere identificata attraverso la ricerca nella colonna GMLID il valore referenziato di *gml:ID*;

2. tutti gli attributi che della tupla identifica devono essere copiati in una nuova tupla, tuttavia gli attributi ROOT_ID e PARENT__ID di questa nuova tupla devono essere settati in accordo alle proprietà della geometria riferita.

Nota

1. se la tupla referenziata è la superiore di una aggregazione (sub) gerarchica all'interno della tabella SURFACE_GEOMETRY poi anche tutte le tuple annidate devono essere copiate in modo ricorsivo e la loro ROOT_ID PARENT_ID devono essere adattati.

2. Copiando le voci già esistenti nella tabella SURFACE_GEOMETRY risulta nelle nuove tuple condividere la stessa GMLID e GMLID_CODESPACE delle originali. Così, questi valori non possono essere utilizzati come chiave primaria.

Quando si tratta di esportare i dati in un documento CityGML, i riferimenti XLink possono essere ricostruiti prendendo traccia dei valori GMLID delle tuple geometriche esportate. Generalmente, per ogni tupla da esportare deve essere verificato se un oggetto geometrico con lo stesso valore di GMLID è già stato elaborato. Se è così, la routine di esportazione dovrebbe fare uso di un riferimento XLink. Tuttavia per verificare il GMLID di ogni singola tupla può rallentare notevolmente il processo di esportazione. Per questa ragione, è stato introdotto l'attributo IS_XLINK in SURFACE_GEOMETRY. Può essere usato per marcare in modo esplicito solo quelle tuple per le quali è necessario che la procedura sia eseguita. Il flag IS_LINK dovrebbe essere usato nella seguente modo. Il tool per l'Importer/Exporter fornisce una corrispondente implementazione

1. **Fase importazione**

   (a) Di default, il flag IS_XLINK è settato a "0".

   (b) Se esistono tuple che devo essere copiate a causa di un riferimento XLink, IS_XLINK viene settato a "1" per ogni copia, e come visto nella nota per ogni copia di tutte le tuple nidificate.

(c) Inoltre, IS_XLINK deve essere impostato su “1” sulla tupla originale riferita dal XLink. Se questa tupla è la radice di una aggregazione (sub)gerarchica, IS_XLINK rimane “0” per tutte le tuple annidate.

2. **Fase esportazione**

   (a) Il processo di esportazione deve soltanto tenere traccia del valore GMLID delle tuple dove IS_XLINK è settato “1”

   (b) Quando viene esportata una tupla con IS_XLINK settato a “1”, il processo di esportazione controlla se ha già incontrato la stessa GMLID e, quindi, può far uso di un referenza XLink nell’istanza del documento.

   (c) Per ogni tupla con IS_XLINK uguale a “0” non si fa nulla.

IS_REVERSE Per quanto riguarda l’attributo IS_REVERSE è anch’esso un campo flag è usato nel contesto di oggetti geometrici gml:OrientableSurface. Generalmente, una istanza di OrientableSurface non può essere rappresentata all’interno della tabella SURFACE_GEOMETRY in quanto non può essere codificata attraverso i flag visti in Tabella 6.2. Tuttavia, il flag IS_REVERSE è usato per poter codificare l’informazione fornita dalla OrientableSurface e rende possibile la ricostruzione in fase di esportazione.

In accordo con GML3, una OrientableSurface consiste di una superficie base e un orientazione. Se l’orientazione è “+”, allora la OrientableSurface e identica alla superficie base. Se l’orientazione è “-”, allora la OrientableSurface è un riferimento ad una superficie con un upnormale, che inverte la direzione per questa OrientableSurface. Vediamo ora come ci si comporta in fase di importazione:

1. Se l’orientazione della OrintableSurface è “-”, allora

   (a) La direzione della superficie base deve essere invertita prima di importarla.(questo significa invertire l’ordine delle coordinate)

   (b) Il flag IS_REVERSE deve essere settato a “1” per la corrispondente entry nella tabella SURFACE_GEOMETRY.

   (c) Se la superficie base è una aggregazione, allora gli step (a) e (b) devono essere ricorsivamente applicati a tutte le superficie appartenenti all’aggregazione.

2. Se l’OrientableSurface è identica alla sua superficie base, allora la superficie base è scritta sulla tabella SURFACE_GEOMETRY senza fare altre operazioni. Il flag IS_REVERSE è posto al valore di default “0”.

3. Si prega di notare che non è sufficiente confidare solo sull’attributo gml:orientation di un OrientableSurface al fine di determinare il suo orientamento in quanto OrientableSurfaces possono essere arbitrariamente annidate.

Nella fase di esportazione usa il flag IS_REVERSE per ricostruire la OrientableSurface nel seguente modo:

1. Se il flag IS_REVERSE e settato a “1” per una entry della tabella, la routine di esportazione deve invertire la direzione del corrispondente oggetto superficie prima di esportarlo(Di nuovo bisogna invertire l’ordine della coordinate).

2. L’oggetto superficie deve essere avvolto da un oggetto gml:OrientableSurface con gml:orentetion’’-”.

3. Se l’oggetto superficie è un aggregato, e le sue superfici appartenenti hanno lo stesso valore per il flag IS_REVERSE non può essere compresa attraverso un’altra OrientableSurface. Tuttavia, se il valore IS_REVERSE cambia, ad esempio da “1” per l’aggregato a “0” per la superficie appartenenti, anche la superficie appartenente deve essere compresa attraverso gml:OrientableSurface in accordo con il punto (2).

come per il IS_XLINK, il tool Importer/Exporter fornisce una implementazione di IS_REVERSE

6.2.2 Appearance Model

La tabella APPEARANCE contiene informazioni sui dati di superficie degli oggetti(attributo DESCRIPTION), la sua categoria è salvata nell’attributo THEME. Per ogni city model o city object e possibile memorizzare i propri dati appearance, la tabella APPEARANCE è in relazione con le tabelle delle classi base CityObject CityModel attraverso due chiavi esterne che possono essere usate alternativamente. La classe Appearance e Surface_data rappresentano caratteristiche, che possono essere referenziate attraverso identificatori GML. Per questa ragione gli attributi GMLID e GMLID_CODESPACE sono stati aggiunti alla tabella.Nella Figura 6.5 è rappresentato parte dello schema riguardante le appearance.

Una appearance è composta da dati per ogni oggetto superficie geometrica. Informazioni sui tipi di dati e le sue appearance sono memorizzati nella tabella SURFACE_DATA. IS_FRONT determina il lato dell’oggetto superficie i dati sono applicati. (IS_FRONT’’1” faccia frontale, IS_FRONT’’0” faccia posteriore). La stringa type denota se il materiale o le texture sono usate per uno specifico oggetto(valore: 3DMaterial, Texture, . . .). Materiali sono specificati attraverso l’attributo X3D_xxx definisce la sua rappresentazione. L’attributo TEX_xxx descrive uso e caratteristiche della texture raster-based 2D. Il link al quale corrisponde le immagini per esempio è specificato dall’attributo TEX_IMAGE_URI. L’immagini delle texture possono essere salvate all’interno della tabella nell’attributo TEX_IMAGE usando il tipo di dato BYTEA di PostgreSQL.

L’attributo GT_XX memorizza tutte le informazioni relative alle texture geo referenziate. La tabella APPEAR_TO_SURFACE_DATA rappresenta l’interrelazione tra le appearance e le superfici.

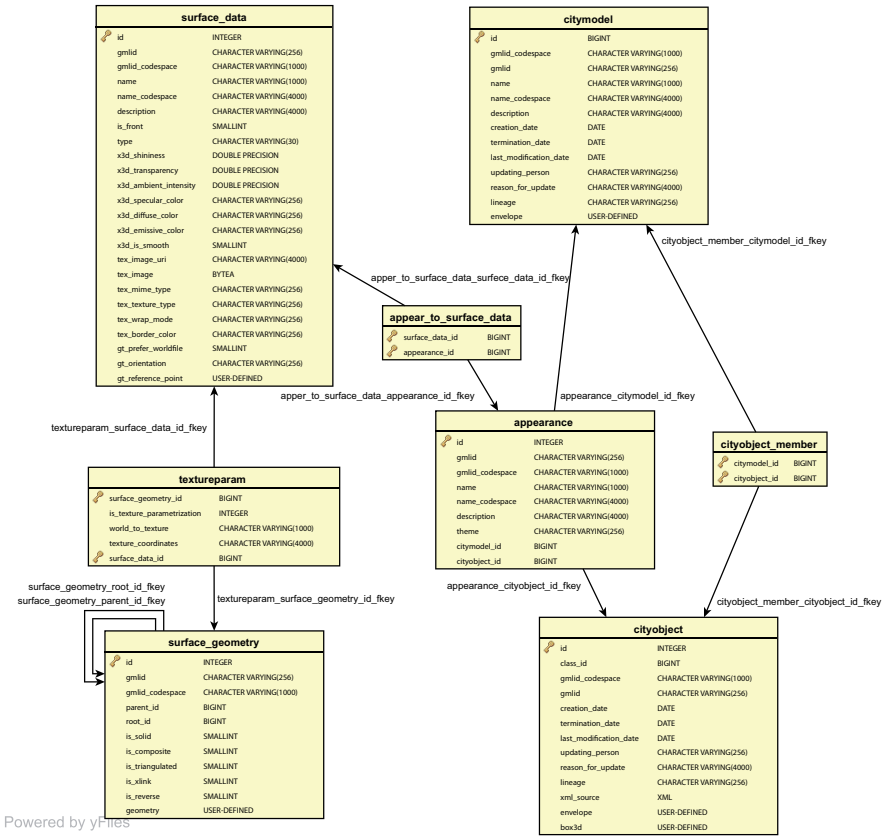

Figura 6.5: schema estratto dal database per l'appearance model

Gli attributi per mappare le texture sugli oggetti (point list o matrici di trasformazione) i quali sono definiti tramite le classi _TextureParameterization, TexCoordList e TexCoordGen di CityGML sono memorizzati nella tabella TEXTUREPARAM.

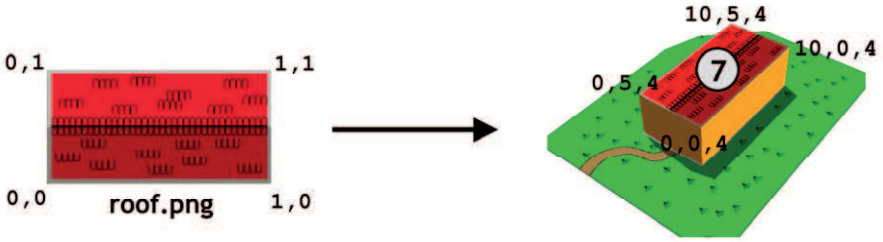

Figura 6.6: Semplice esempio che rappresenta come mappare una texture con le coordinate

Le coordinate delle texture sono applicate alle superficie dei poligoni, i cui contorni sono descritti attraverso una linear ring chiusa. Le coordinate sono salvate in una

SURFACE _GEOME-TRY	IS_TEXTURE _PARAMETR IZATION	WORLD_TO _TEXTURE	TEXTURE _COORDI-NATES	SURFACE _DATA
10	1	NULL	0.0 0.0 1.0 0.0 1.0 1.0 0.0 1.0 0.0 0.0	20
…	…	…	…	…

Tabella 6.3

stringa con una lista di valori decimali separati da spazi bianchi. L'attributo WORLD_ TO_ TEXTURE definisce una matrice di trasformazione.

6.2.3 Thematic Model(Building Model)

Il modello Building descritto in precedenza a livello concettuale, è realizzato attraverso le tabelle mostrate in Figura 6.7.

Le tre classi CityGML *AbstractBuilding*, *Building* e *BuildingPart* sono fuse nella singola tabella BUILDING. La relazione di sottoclasse con CITYOBJECT si pone utilizzando gli stessi ID, per ogni tupla di BUILDING deve esistere una tupla in CITYOBJECT con lo stesso ID.

Il significato e il nome di molti campi è identico a quelli degli attributi del diagramma UML. La geometria è rappresentata attraverso quattro chiavi esterne LoD1_ GEOMETRY_ ID a LOD4_ GEOMETRY_ ID che riferiscono alle etries nella tabella SURFACE_ GEOMETRY e rappresentano ogni livello di dettaglio geometrico LoD.

La componente gerarchica appartenente a un building è realizzata attraverso la chiave esterna BUILDING_PARENT_ID che si riferisce al building superiore (aggregato) e contiene NULL, se tale non esiste. In questo modo abbiamo una struttura ad albero anche per i building come per gli aggregati. BUILDING_PARENT_ID punta al predecessore nell'albero. La chiave esterna BUILDING_ROOT_ID si riferisce direttamente al livello più alto (radice) di un albero di building. Al fine di selezionare tutte le parti che formano un unico building non ha che da scegliere quelli con la stessa BUILDING_ROOT_ID.

Opzionalmente la geometria di selezione della curva di intersezione del terreno è salvata nell'attributo LODx_TERRAIN_ITERSECTION($1 \leq x \leq 4$) come MULTILINESTRING (MultiCurve). Inoltre elementi lineari di building, quali antenne, possono essere memorizzate nel campo LODx_MULTI_CURVE ($1 \leq x \leq 4$), anche utilizzato in SDO_GEOMETRY (MultiCurve)).

Le opening (CityGML *Opening* CityGML, aperture ad esempio porte e finestre) sono rappresentate attraverso la tabella OPENING. Non è stata realizza una tabella per ogni singola sottoclasse. Invece la differenziazione è realizzata attraverso l'attributo TYPE nella tabella OPENING. La tabella OPNENING_TO_THEM_SURFACE associa un ID opening nella tabella OPENING con l'ID di una superficie tematica nella tabella THEMATIC_SURFACE rappresentando la relazione m:n tra entrambe le tabel-

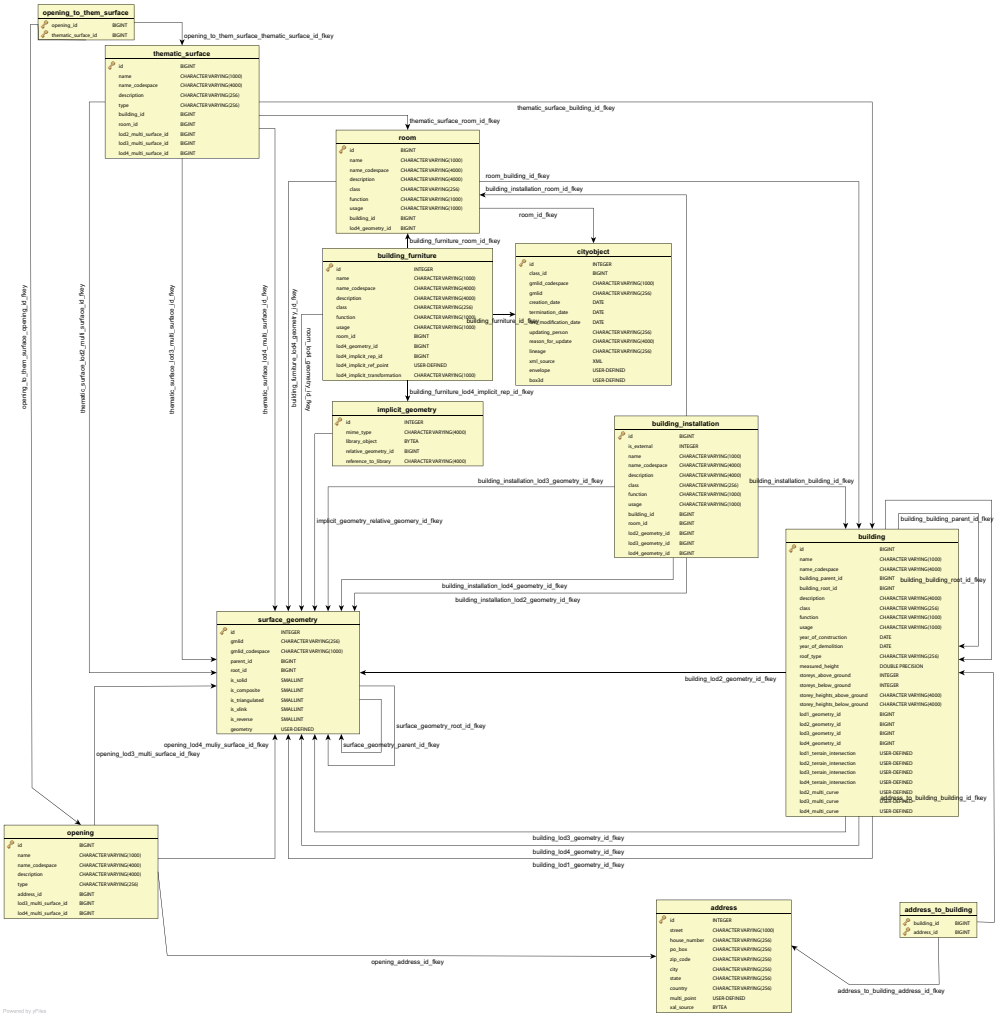

Figura 6.7: schema estratto dal database per il building model

le. Un indirizzo può essere assegnato ad una porta (tabella OPENING) attraverso la chiave esterna ADDRESS_ID nella tabella OPENING. Inoltre, l'indirizzo può essere assegnato al building (vedere la tabella ADDRESS).

la relazione di aggregazione tra building e le corrispondenti superficie perimetrali è data dalla chiave esterna BUILDING_ID della tabella THEMATICH_SURFACE che riferisce l'ID del rispettivo building. Le thematic surface e i corrispondenti building dovrebbero condividere la loro geometria: la geometria dovrebbe essere definita una sola volta e può essere usata congiuntamente come XLink.

Le classi UML *BuildingIstallation* e *IntBuildingIstallation* sono realizzate attraverso la singola tabella BUILDING_INSTALLATION. Oggetti interni ed esterni sono distinti attraverso l'attributo IS_EXTERNAL. La relazione con il corrispondente building avviene attraverso la chiave esterna BUILDING_ID, mentre la geometria in LoD

2 a 4 è data attraverso la chiave esterna LODx_GEOMETRY_ID ($2 \leq x \leq 4$) riferita alla tabella SURFACE_GEOMETRY.

L'oggetto *room*(stanze) è permesso solo in LOD 4. Pertanto la sola chiave LOD4_ GEOMETRY_ ID è riferita alla tabella SURFACE_GEOMETRY. Inoltre le chiavi esterne alle tabelle BUILDING e CITYOBJECT sono necessarie per mappare la relazione con queste tabelle.

Nelle stanze ci possono essere elementi di arredo(sedie armadi, ...), una chiave esterna che referenzi ROOM_ID è obbligatoria. La geometria di oggetti *furniture*(arredo) può essere descritta esplicitamente usando la chiave esterna LOD4_ GEOMETRY_ ID o tramite l'uso di prototipi che sono memorizzati come una libreria di oggetti. Le informazioni necessarie per mappare gli oggetti prototipo delle rooms consiste di un punto base e una matrice di trasformazione salvati negli attributi LOD4_ IMPLICIT_ REF_ POINT e LOD4_ IMPLICIT_ TRASFORMATION.

Gli indirizzi sono realizzati tramite la tabella ADDRESS. La relazione m:n viene sciolta con la tabella ADDRESS_TO_BUILDING che associa un building ID e un address ID. Come visto in precedenza un indirizzo può essere associato anche ad una porta. IL successivo ID disponibile per la tabella ADDRESS è fornito dalla sequenza ADDRESS_SEQ.

6.2.4 CityFurniture Model

Gli attributi della classe *CityFurniture* specificata nel diagramma UML di CityGML sono direttamente mappati nella tabella CITY_FURNITURE rappresentata nella Figura 6.8.

La geometria degli oggetti furniture è rappresentata sia come un oggetto (LODx_ GEOMETRY_ ID ($1 \leq x \leq 4$)) in relazione con la tabella SURFACE_GEOMETRY o come geometria implicita. Nel caso di una geometria implicita, deve essere data una referenza ad un oggetto di tipo point (LODx_IMPLICIT_REF_POINT ($1 \leq x \leq 4$)) e opzionalmente con una matrice di trasformazione (LODx_ IMPLICIT_ TRASFORMATION ($1 \leq x \leq 4$)). In ordine computare l'attuale localizzazione dell'oggetto, deve essere processata la trasformazione delle coordinate locali nel sistema di riferimento del modello della città e le coordinate del punto di ancoraggio devono essere aggiunte.

6.2.5 Digital Terrain Model

Una tupla nella tabella RELIEF rappresenta un oggetto complesso *relief*(rilievo), che consiste di differenti componenti relief. Ha un attributo LODGROUP che descrive il legame tra un oggetto relief e un certo livello di dettaglio(LoD). L'attributo NAME contiene il nome dell'oggetto. Le singole componenti di un oggetto relief complesso sono memorizzate nelle tabelle BREAK_RELIEF, MASSPOINT_RELIEF, TIN_RELIEF, e RASTER_RELIEF. Ogni componente relief ha un proprio LoD. Tuttavia ogni singolo componente di un oggetto relief complesso può appartenere ha differenti LoD e essere eterogeneo. La separazione geometrica tra i singoli componenti di un oggetto relief complesso è realizzata tramite poligoni(attributo EXTENT), che specifica l'area

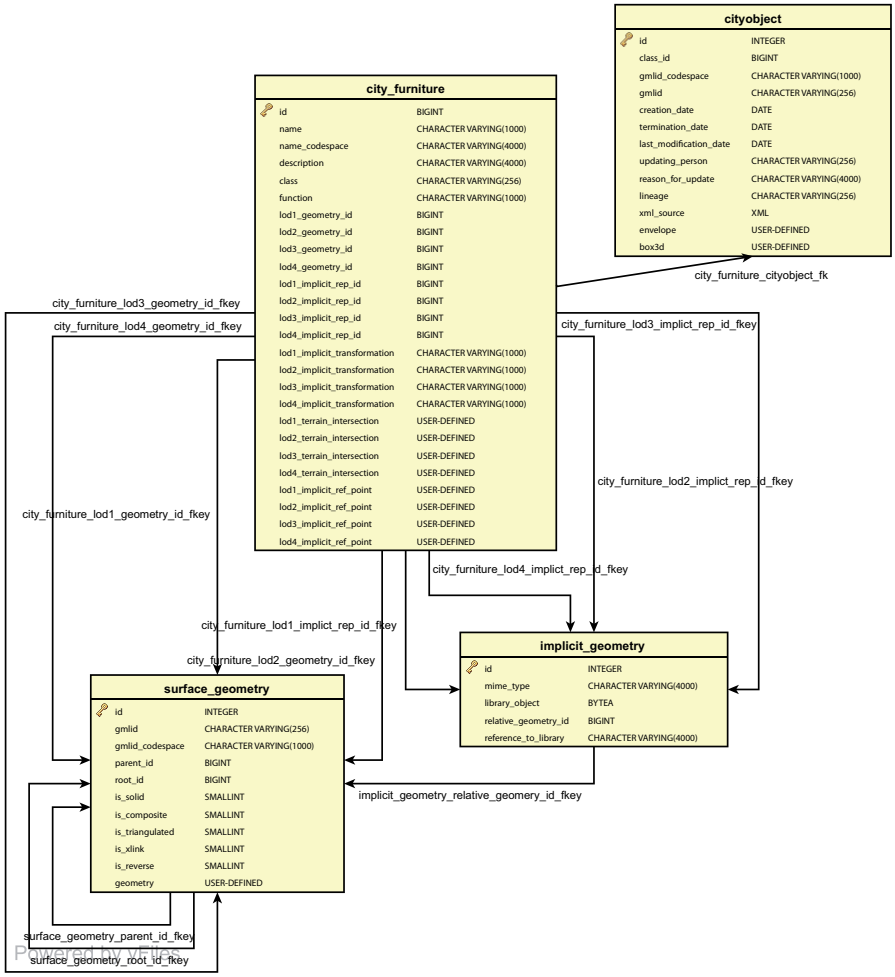

Figura 6.8: schema estratto dal database per il furniture model

valida del componente relief. Ogni componetene relief ha un attributo NAME che è usato per nominare le componenti. IL relief cosi come ogni componente relief deriva dalla tabella CITYOBJECT e riceve lo stesso ID come il CityObject. La tabella RELIEF_FEAT_TO_COMP rappresenta la interrelazione tra le caratteristiche rielief e i componenti relief. Nella Figura 6.9 è rappresento parte dello schema che riguarda la classe relief.

6.2.6 Generic CityOject Model

Un modello City 3D molto probabilmente contiene attributi, che non sono esplicitamente modellati in CityGML. Inoltre, essi possono essere oggetti 3D che non sono coperti dalle classi tematiche di CityGML. Oggetti e attributi generici aiutano a supportare la memorizzazione di questi dati.

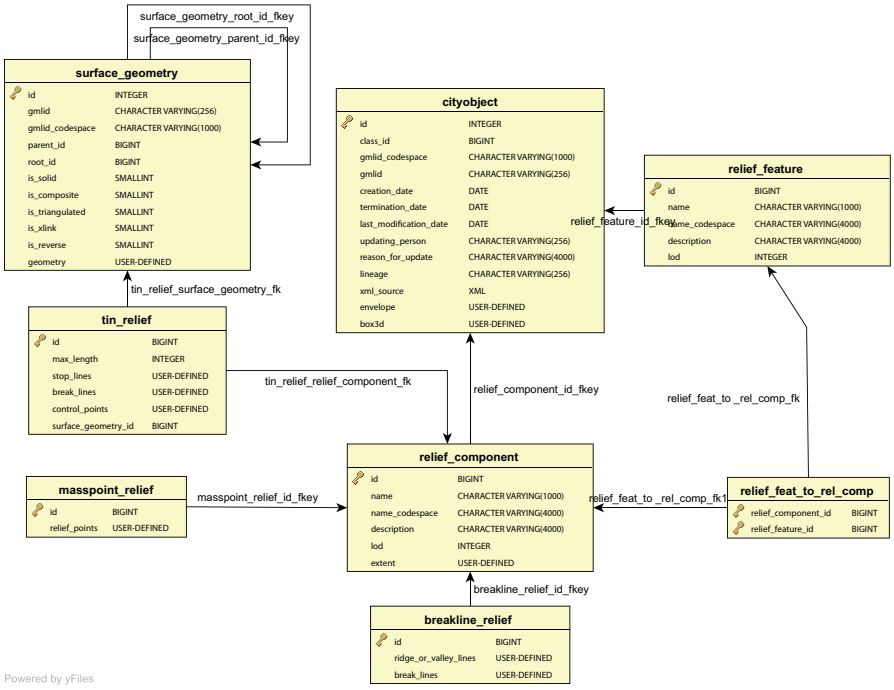

Figura 6.9: schema estratto dal database per il Digital Terrain Model

La tabella GENERIC_CITYOBJECT è la tabella principale di questa parte dello schema rappresentato in Figura 6.10. Le informazioni geometriche di un generico oggetto cityobject possono essere memorizzate utilizzando coordinate assolute del mondo come una surface geometry (LODxGEOMETRY_ID ($1 \leq x \leq 4$)) o come geometria implicita. Nel caso si usi la geometria implicita, deve essere dato un punto di riferimento dell'oggetto (LODx_ IMPLICIT_ REF_ POINT ($1 \leq x \leq 4$)), e opzionalmente una matrice di trasformazione (LODx_IMPLICIT_ TRASFORMATION ($1 \leq x \leq 4$)). In ordine computare l'attuale localizzazione dell'oggetto, deve essere processata la trasformazione delle coordinate locali all'interno del sistema di riferimento del modello city e aggiunte le coordinate del punto di ancoraggio (LODx_ IMLPLICIT_ REP_ID ($1 \leq x \leq 4$)).

Inoltre, per specificare l'intersezione esatta del DTM con la geometria 3D di un *GenericCityObject*, il *TerrainIntersectionCurve* può essere aggiunto per ogni LoD (LODxTERRAIN_ INTERSECTION ($1 \leq x \leq 4$)).

La tabella CITYOBJECT_GENERICATTRIB è usata per rappresentare il concetto generico di attributo. Però, la creazione di una tabella per ogni tipo di attributo è stata omessa. Invece si utilizza una singola tabella per rappresentare tutti i tipi, i tipi sono differenziati attraverso il valore dell'attributo DATATYPE come mostrato nella Tabella 6.4.

Notiamo che il datatype BLOB non può essere gestito in PostgreSQL per questo non viene utilizzato, La relazione tra il generico attributo e il corrispondente CityObject

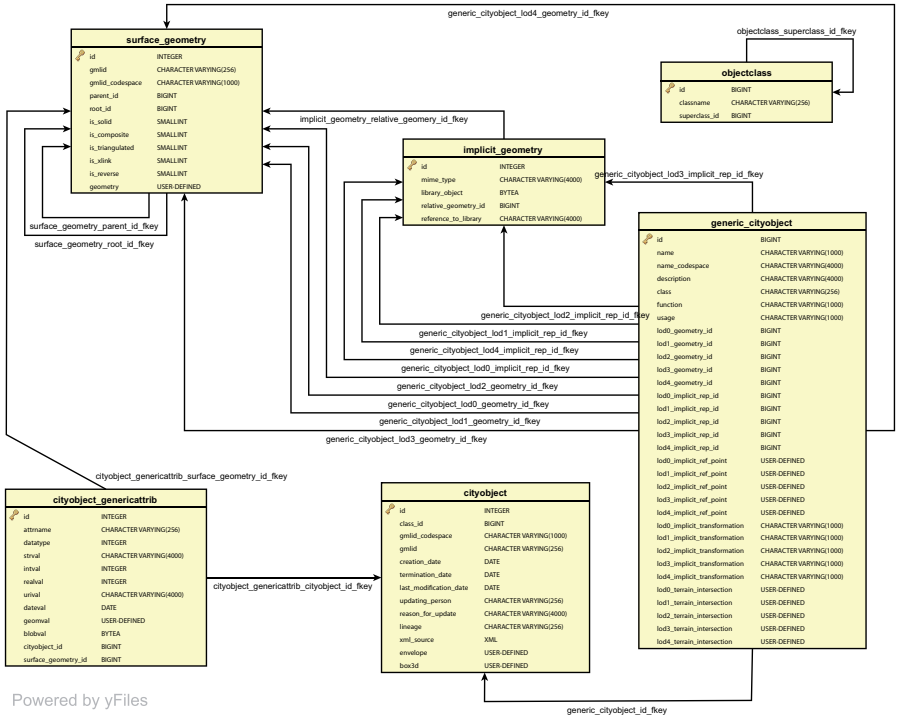

Figura 6.10: schema estratto dal database per il Generic CityObject Model

è stabilita tramite la chiave esterna CITYOBJECT_ID (REL_CITYOBJ_ID_ID_1).

Il successivo ID disponibile per la tabella CITYOBJECT_GENERICATTRIB è fornito tramite la sequenza CITYOBJECT_GENERICATT_SEQ.

Nella tabella OBJECTCLASS elenca tutti i nomi delle classi dello schema. La relazione di sottoclasse con la classe genitore è rappresentata tramite l'attributo SUPERCLASS_ID, nella sottoclasse come un chiave esterna del ID della classe genitore.

DATATYPE	**Attribute Type**
1	STRING
2	INTEGER
3	REAL
4	URI
5	DATE
6	BLOB(non utilizzato in PostgreSQL)
7	Geometria PostGIS
8	Geometria attraverso una superficie nella tabella SURFACE_GEOMETRY

Tabella 6.4: Tipo di attributo di CITYOJECT_GENERICATTRIB

6.2.7 LandUse Model

La tabella LAND_USE contiene tutti gli attributi specificati nel diagramma UML di CityGML. La relazione con la tabella SURFACE_GEOMETRY è stabilità attraverso la chiave esterna LODx_ MULTI_ SURFACE_ ID ($1 \leq x \leq 4$). Nella Figura 6.11 è rappresentato lo schema del modello LandUse.

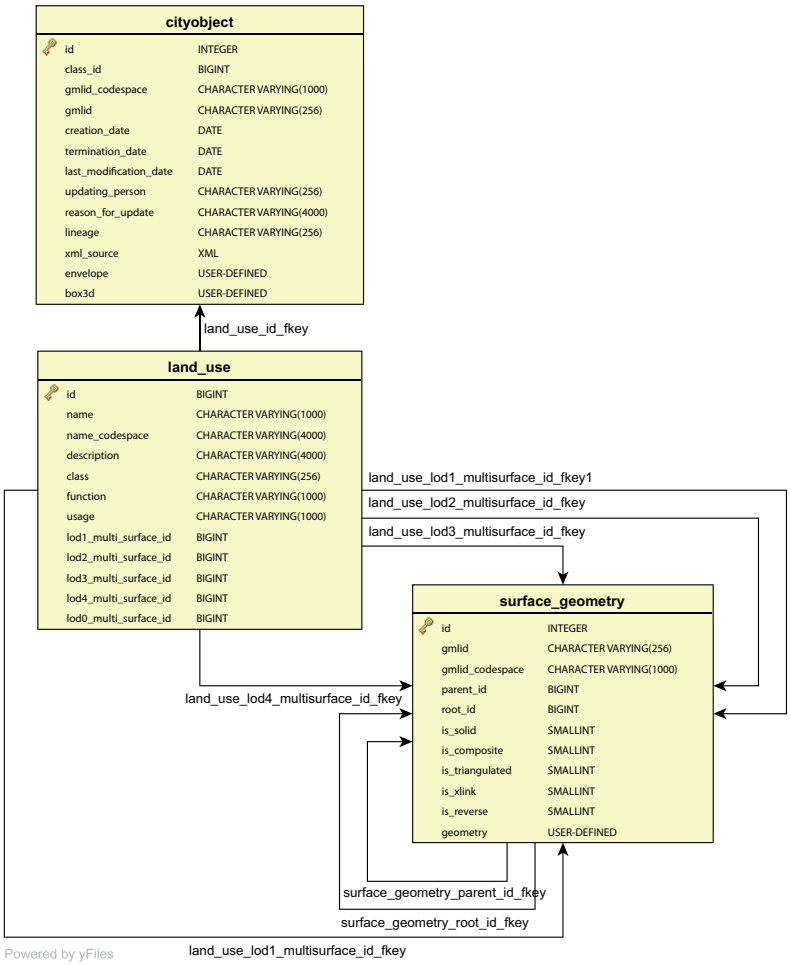

Figura 6.11: schema estratto dal database per il LandUse Model

6.2.8 Transportation Model

Per la realizzazione degli oggetti transportation sono necessarie due tabelle, TRAFIC_ AREA e TRANSPORTATION_ COMPLEX.

La tabella TRAFFIC_AREA implementa l'omonima classe UML e la classe *AuxliaryTrafficArea* attraverso l'attributo IS_AUXILIARY, che rappresenta una *TrafficArea*

con valore “0” o una *AuxliaryTrafficArea* con valore pari a “1”. La rappresentazione degli oggetti geometrici è gestita tramite la chiave esterna LODx_ MULTI_ SURFACE_ ID ($2 \leq x \leq 4$).

Così come mostrato nel diagramma UML, ogni oggetto della classe *TrasportationComplex* è mappato nella tabella TRASPORTATION_COMPLEX, può avere gli attributi *function* e *usage*. Per differenziare le sottoclassi *Track*,*Road* e *Square* si usa l’attributo TYPE. Ad un livello di dettaglio basso, gli oggetti sono modellati come linee, LOD0_NETWORK di tipo MILTISTRING. A partire dal livello di dettaglio “1” la rappresentazione degli oggetti geometrici è data dalle chiavi esterne LODx_ MULTI_ SURFACE_ ID ($1 \leq x \leq 4$). Nella Figura 6.12 è rappresentato lo schema del Transportation model.

Figura 6.12: schema estratto dal database per il Transportation Model

6.2.9 Vegetation Model

Il vegetation model è realizzato tramite le tabelle mostrate nella Figura 6.13 che corrisponde in gran parte al modello UML.

Nella tabella SOLITARY_VEGETAT_OBJECT gli attributi *class*, *species*, *function*, *height*, *trunkDiameter* e *crownDiameter* descrivono una singolo oggetto vegetazione. Le informazioni geometriche possono o essere memorizzate con coordinate assolute mondiali con alcune surface geometry (LODx GEOMETRY_ ID ($1 \leq x \leq 4$)),

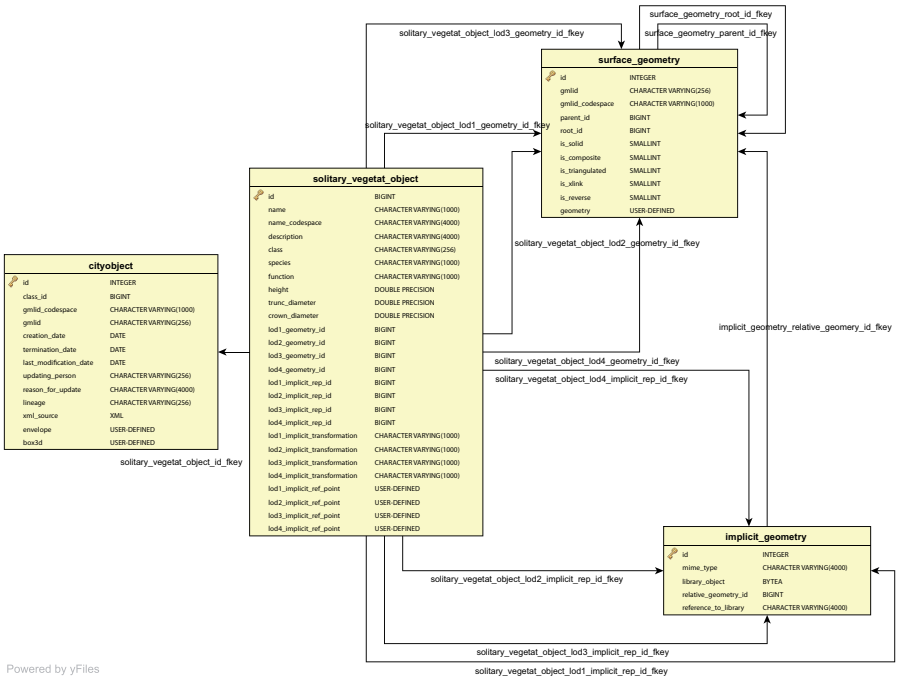

Figura 6.13: schema estratto dal database per il Vegetation Model

o come geometria implicita. Nel caso di geometria implicite ci si comporta come visto in precedenza.

Nella tabella PLANT_COVER contiene informazioni sulla zona di vegetazione negli attributi *class*, *function* e *averageHeight*. La geometria è ristretta ad una MultiSurface o MultiSolid e sono rappresentate tramite la chiave esterna LODxGEOMETRY_ID ($1 \leq x \leq 4$) che riferisce alla tabella SURFACE_GEOMETRY.

6.2.10 WaterBody Model

La modellazione del WATERBODY(Corpi Idrici) nello schema del database Figura 6.14, corrisponde in gran parte al suo rispettivo modello UML (*class, function, usage*). Per LoD0 e LoD1 sono aggiunti attributi opzionali ad esempio per modellare la geometria delle rive (LODx_MULTI_CURVE). La geometria LoD0 e LoD1 delle aree dei corpi idrici è memorizzata nella tabella SURFACE_GEOMETRY. La chiave esterna LODx_MULTI_SURFACE_ID ($1 \leq x \leq 1$) riferisce alla corrispondente riga nella tabella SURFACE_GEOMETRY. La geometria che rappresenta volumi pieni d'acqua è gestita in maniera simile usando la chiave esterna LODx_SOLID_ID ($1 \leq x \leq 1$). Per mappare l'aggregazione *boundedBy* che identifica il rivestimento esterno del corpo idrico gestito tramite la tabella WATERBOUNDARY_SURFACE e l'addizionale tabella WATERBOD_TO_WATERBND_SRF necessaria per realizzare la relazione n:m tra le due classi.

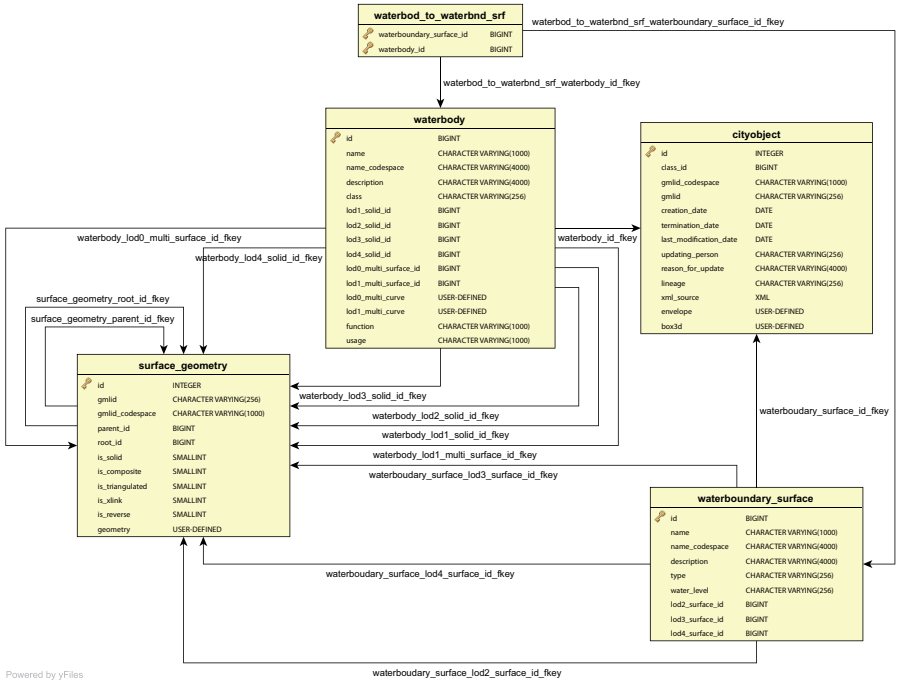

Figura 6.14: schema estratto dal database per il WaterBody Model

Il WATERBODY(corpo idrico) può essere differenziato semanticamente tramite la classe *WaterBoundarySurface*, che è realizzata tramite le caratteristiche delle classi *WaterSurface*, *WaterGroundSurface* o *WaterClousureSurface*. La classe è realizzata attraverso la tabella WATERBOUNDARY_SURFACE e la differenziazione è gestita tramite l'attributo TYPE. Dato che ogni oggetto WaterBoundarySurface deve avere almeno una surface geometry associata le chiave esterne LODx_SURFACE_ID ($1 \leq x \leq 4$) sono usate per realizzare questa relazione.

6.2.11 Sequenze, Database_SRS

La tabella DATABASE_SRS è usata per definire il sistema di riferimento spaziale per un certo CityModel. Contiene solo una tupla con gli attributi SRID che identifica il sistema di riferimento utilizzato all'interno della base dati e il nome corrispondente CRS. L'informazione è obbligatoria e viene inserita durante l'installazione dello schema del database. SRID descrive l'identificativo del riferimento spaziale, e GML_SRS_NAME l'identificatore appropriato per GML (usato nell'attributo *srsName*).

Nella Figura 7.1 sono definite delle sequenze che sono usate per generare numeri sequenziali, per definire automaticamente chiavi primarie uniche. Sono sequenze ascendenti che partono da un valore pari a 1, e vengono via via incrementate di una unità alla volta. Le sequenze inserite sono le seguenti: ADDRESS_SEQ, APPEARANCE_SEQ, IMPLICT_ GEOMETRY_SEQ, CITYOBJECT_SEQ, SURFACE_ GEO-

METRY_SEQ, CITYMODEL_SEQ, SURFACEDATA_SEQ, RASTERRELIEF_ IMP_ SEQ, EXSTERNAL_SEQ, CITYOBJECT_ GENERIC_ ATT_SEQ.

Capitolo 7

ToolKit di Importazione e Esportazione

In questa sezione si descrive il software sviluppato per l'importazione e l'esportazione dei dati rappresentati secondo lo schema definito. Ricordiamo che per ottimizzare le prestazioni si utilizza una architettura software multithread. La Figura 7.1 riporta lo schema dell'architettura adottata (in perfetta analogia con il progetto IGGS Technische Universität Berlin *3D City DB* [3]). Come si può vedere si identificano tre parti, nel mezzo il processo di *binding* che crea un modello di oggetti JAVA dalla definizione dello schema XSD di CityGML, la parte superiore mostra il processo di importazione mentre quello inferiore quella di esportazione

Particolare attenzione è stata posta nella gestione degli ID degli oggetti, perché solo l'uso di un identificativo univoco a livello mondiale, è garanzia di un coerente aggiornamento degli oggetti geografici memorizzati nel database. Considerato che i GMLID sono opzionali e devono essere unici all'interno di dataset, o a tutti gli oggetti importati viene assegnato un nuovo identificatore unico al mondo (UUID), o solo agli oggetti ai quali manca il GMLID viene assegnato un UUID. In entrambi i casi, l'attributo GMLID_CODESPACE è aggiunto ad ogni oggetto(incluse superficie e geometrie di base).

7.1 Progettazione software del tool di Impot/Export

Il tool è implementato come una applicazione JAVA. Impiega una strategia di trasformazione dei documenti XML un pezzetto alla volta, in modo da gestire documenti di dimensioni arbitrarie.

7.1.1 Supporto per istanze di documenti CityGML di grande dimensioni

L'elaborazione di grandi documenti è realizzato attraverso uno stream-base XML al quale vengono associati degli oggetti JAVA. Questo approccio utilizza due framework di Java esistenti per la lettura e scrittura di file XML prolissi. Da un lato l'architettura

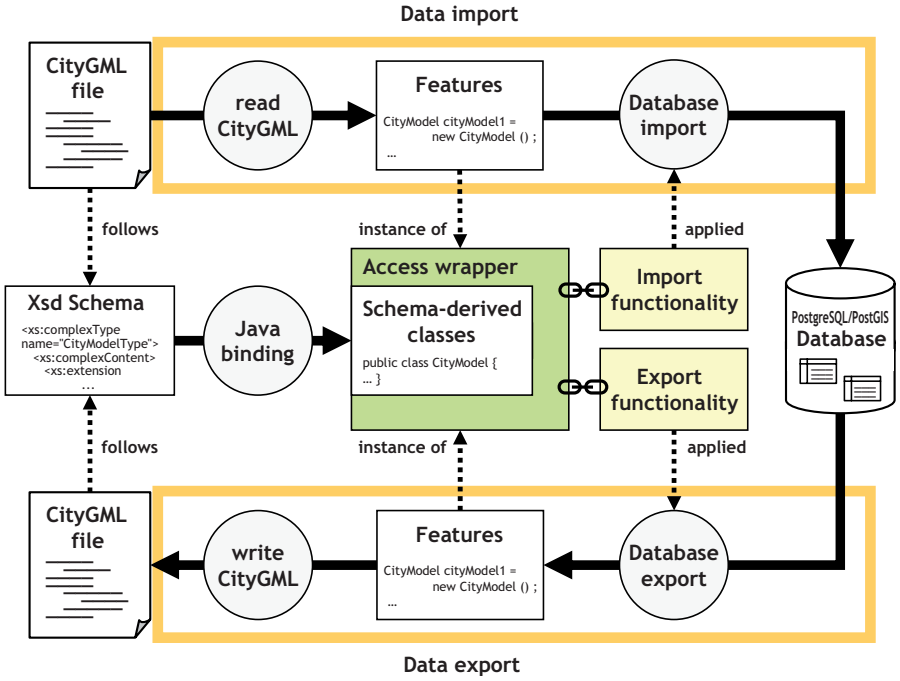

Figura 7.1: Tool di import/export: panoramica del flusso globale dei dati

JAVA XML Binding (JAXB) per facilitare una visione orientata ad oggetti di dati XML. D'altra parte, Simple API per XML (SAX) viene utilizzata per lo **stream-based** e *event-driver parsing* di un documento XML. Combinando entrambi i framework si possono sfruttate i loro benefici, limitando i loro lati negativi.

Il framework JAXB fornisce una via conveniente e facile all'uso per validare ed elaborare i documenti XML. Java Architecture for XML Binding (JAXB) permette agli sviluppatori Java di effettuare il mapping tra le classi JAVA e una loro corrispondente rappresentazione XML. JAXB fornisce la possibilità di serializzare oggetti Java in XML (marshalling) e di effettuare l'operazione inversa (unmarshalling), ovvero permette di ottenere a partire da un file XML la corrispondente rappresentazione a oggetti Java. JAXB permette quindi di manipolare file XML senza la necessità di implementare alcuna routine specifica per il salvataggio e la lettura di dati. Grazie a queste API possiamo creare dei veri e propri oggetti Java a partire dalla definizione del file XML, definizione che viene di solito fatta usando DTD o XML Schema. A partire da queste definizioni vengono create dinamicamente delle classi JAVA che poi conterranno i dati riportati nel file XML. JAXB richiede che lo stream o l'intero file del documento XML sia presente nella memoria centrale prima di elaborare i dati. Questo può diventare un problema nel caso in cui i documenti XML siano di grandi dimensioni. Al contrario, i *parser streaming* come ad esempio SAX, permettono un accesso seriale ai documenti XML. Ogni elemento del documento è passato all'applicazione in ordine della sua comparsa attraverso i metodi di callback definiti dall'utente. Di conseguenza,

l'ingombro di memoria di questo approccio *event-driven* si basa principalmente sui dati memorizzati in un singolo elemento XML, ed è quindi sempre solo una piccola frazione della dimensione di tutto il documento. Tuttavia, l'analisi è basata su un flusso di dati XML unidirezionale, così che dati a cui si è acceduto in precedenza non possono essere riletti senza la rielaborazione dell'intero documento. Al fine di fornire sia una visione orientata agli oggetti per dati XML e una soluzione per il problema del sovraccarico dei dati, il tool di import/export implementa in due fasi l'approccio visto:

1. I documenti XML vengono parsati utilizzando un SAX *streamming parser* che divide l'intero documento in pezzi di dimensioni gestibili. Per ogni blocco, sono registrati utilizzando un buffer di memoria.

2. Il contenuto del buffer viene individualmente mappato in oggetti JAVA per l'elaborazione dei dati usando JAXB. Per la scrittura dei documenti XML viene applicato il processo inverso

In questo modo, l'utilizzo della memoria non è più dipendente dalla dimensione del documento, ma si riferisce solo alla quantità di dati conservati nel buffer di memoria. Per i documenti CityGML ad esempio, è ragionevole che un blocco sia pari al contenuto presente tra due tag XML <cityObjectMember> che rappresentano le feature ad alto livello all'interno di un modello CityGML.

7.1.2 Concorrenza e elaborazione dei dati

L'architettura di base dell'applicazione sviluppata si basa sull'esecuzione concorrente di più task. Generalmente, ogni task del processo di importazione o esportazione è eseguito da un diverso thread. L'approccio "un thread per ogni task", però, presenta alcuni svantaggi se c'è un numero troppo elevato di thread attivi. Per questo, l'applicazione, riusa i thread per diversi task dello stesso tipo. In questo modo il costo per la creazione dei thread è distribuito tra i diversi task, e dato che il thread esiste già quando un task è richiesto, il ritardo introdotto dalla creazione del thread è eliminato.

L'insieme dei thread è implementato come una coda di compiti (work) abbinato ad una coda di worker thread, vedi Figura 7.2. La prima coda (work queue), è realizzata come una coda bloccante, paradigma tipico nella programmazione concorrente: un thread produce oggetti, ad esempio task da eseguire, e li inserisce in una coda affinché questi siano processati da un altro thread, che li rimuove dalla coda. La coda bloccante segue il paradigma del produttore-consumatore, in modo da evitare deadlocks.

Singoli insiemi di thread possono essere combinati per completare l'intera catena produttiva. I worker thread all'interno di un insieme processano contemporaneamente i task che si trovano nella coda dei worker thread associata, e passa i risultati alla coda successiva. Questo permette il disaccoppiamento delle fasi dei processi in modo modulare, permettendo così una facile estensione per altre fasi del processo.

Le comunicazioni tra le vari fasi del processo avviene attraverso un thread di tipo event-dispatcher che implementa un singolo worker pool, permettendo uno scambio di messaggi sincrono ed asincrono, tipico del paradigma produttore-consumatore.

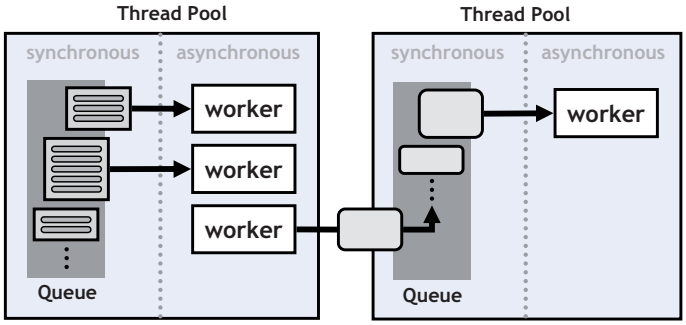

Figura 7.2: Thread poll implementato come coda di work con gruppi fissati di worker thread

Il numero ottimale di "worker thread" all'interno di un pool di thread dipende principalmente dal numero di processori disponibili, dalla natura del task da eseguire e dal costo del cambio di contesto dei thread. Il numero ottimale di "thread worker" all'interno di un pool di thread dipende principalmente dal numero di processori disponibili, la natura del compito, per esempio, legati alle operazioni I/O o legati all'elaborazione, e al costo del passaggio di contesto del thread. Il tool di import/export mantiene i valori di soglia definiti dall'utente e di una unità di gestione per thread pool per l'adeguamento autonomo in base al carico di lavoro complessivo. Inoltre, la dimensione della coda è un fattore determinante per il consumo di memoria, e quindi può essere regolata per specifiche configurazioni del sistema.

7.2 Importazione

7.2.1 Processo d'importazione

Il processo di importazione del dataset CityGML all'iterno del database è illustrato nella Figura 7.3. Il flusso di lavoro è implementato dal concatenamento di un pool di thread che coprono ogni singolo step del processo d'importazione. Questo comprende l'insieme di thread per l'elaborazione a pezzi(chunk) del documento XML d'ingresso, (pool di parser thread), la conversione dei singoli pezzi(chunk) in oggetti JAXB che rappresentano oggetti CityGML, (poll di converter thread), e l'elaborazione di questi oggetti, (pool di import e XLink thread).

Il primo ed il secondo passo della catena dei processi, cioè, elaborazione del *chunk* di dati in input e l'unmarshalling delle caratteristiche top-level di CityGML, sono già state discusse in precedenza. Analizzando gli oggetti JAXB risultanti, la sottosequenza di "importer thread" illustrata in Figura 7.3, ricava le istruzioni SQL finali, utilizzando le funzioni di importazione, le quali mappano gli oggetti con le corrispondenti tabelle del database relazionale. Questo passo può essere castomizzato attraverso un filtro definito dall'utente. Per esempio, l'importazione può essere limitata ad un dato insieme di tipi di caratteristiche, ad esempio solo edifici CityGML o di essere limitato in una

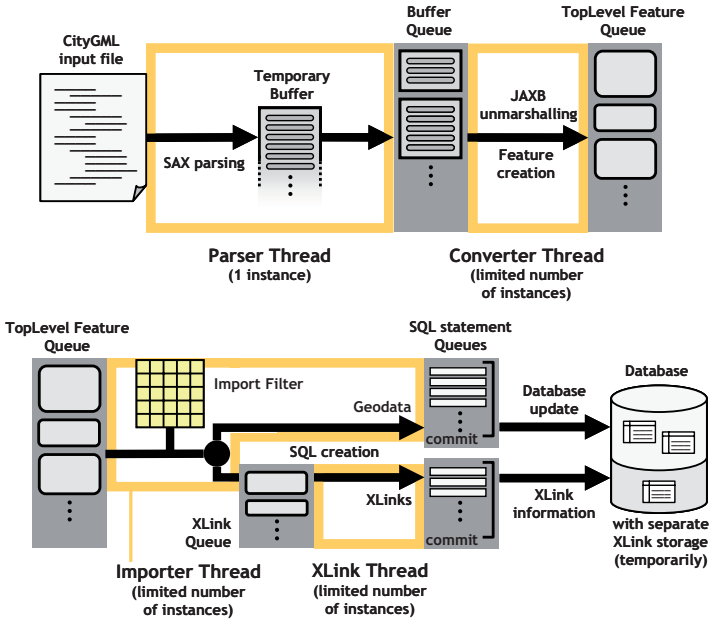

Figura 7.3: flusso di lavoro dettagliato dall'ingresso di una istanza di un documento CityGML all'archiviazione nel database. I riferimenti XLink sono memorizzati separatamente nelle tabelle temporanee per poter risolvere gli XLink nella fase 2.

certa area "Bounding Box". Gli oggetti filtrati possono essere immediatamente saltati e rilasciati dalla memoria.

In modo tale da ottimizzare i tempi di risposta del database, le istruzioni SQL vengono precompilate e memorizzate nei corrispondenti oggetti Java dagli "import thread". Le istruzioni precompilate possono essere riutilizzate per utilizzarle più volte durante l'esecuzione con dati diversi. Inoltre, le istruzioni SQL sono collezionate e inoltrate al database solo dopo aver raggiunto un certo numero di istruzioni.

L'architettura multithreading permette l'esecuzione concorrente di tutte gli step del processo di importazione. Ad esempio, il parsing di un documento XML non è mai bloccato da un thread in attesa di una risposta dal database, e più oggetti JAXB possono essere elaborati simultaneamente. Comunque, l'elaborazione a pezzi(chunk) dei file di input introduce un aumento della complessità. Nei documenti CityGML, le proprietà di alcuni oggetti possono essere espresse sotto forma di XLink. Quindi, invece di avere il contenuto XML a disposizione con l'elemento estratto, questo è dato da una referenza ad un oggetto remoto identificato col suo GMLID. Fin tanto che le referenze all'interno dello stesso documento XML sono permesse, per risolvere gli XLink un approccio di tipo "lineare" non è sufficiente, in quanto richiederebbe di avere l'intero documento in memoria.

Per risolvere questo problema si adotta una strategia a due fasi. Nella prima eseguita Figura 7.3, gli oggetti sono scritti nel database ignorando eventuali referenze ad oggetti remoti. Se un oggetti contiene uno XLink, si scrive una tupla aggiuntiva in una

tabella temporanea da un thread separato (XLink thread). Questa tupla contiene, principalmente, l'oggetto che contiene lo XLink e il GMLID dell'oggetto riferito. Inoltre, come già spiegato precedentemente, l'applicazione tiene memoria dei GMLID degli oggetti già incontrati e la corrispondente rappresentazione nel database. In un secondo momento, solo queste le tabelle temporanee sono richieste e riviste dall'applicazione. Poiché a questo punto l'intero documento è stato elaborato, le referenze valide possono essere risolte e di conseguenza processate. La seconda fase del processo di importazione è mostrato nella Figura 7.4:

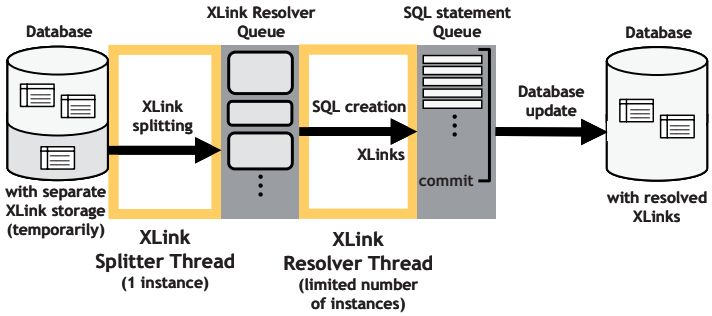

Figura 7.4: Dopo che la fase 1 è completata, le informazioni memorizzate nelle tabelle temporanee vengono interrogate (pool splitter thread) per risolvere i riferimenti XLink attraverso oggetti remoti(pool di resolver thread).

7.3 Esportazione

7.3.1 Processo di esportazione d'istanze di documenti CityGML

IL workflow per l'esportazione del dataset CityGML dal database è illustrata nella figurename 7.5. Come per l'importazione, il workflow è realizzato come una catena di processi che unisce diversi pool di thread, ognuno dei quali ricopre un diverso step del processo.

Il primo step di questo workflow è quello di interrogare il database in base ai criteri definiti dall'utente. Questi filtri di esportazione sono definiti in maniera identica hai filtri d'importazione, ad esempio, permettendo interrogazioni spaziali all'interno di una data area(bounding box). Inoltre, le interrogazioni sono solamente eseguite su una singola tabella del database nella prima esecuzione, la quale contiene soltanto informazioni generali per tutte le feauture contenute. Così, le interrogazioni possono essere rapidamente processate nel database e ritornate dal tool di esportazione.

Non appena della prima interrogazione sono restituiti, lo "splitter thread" li inserisce nella *worker queue* della sottosequenza di "export thread". In questo modo, il "worker thread" del "pool export" ha già iniziato a ricostruire i corrispondenti oggetti CityGML, che i "splitter thread" non hanno ancora elaborato la risposta iniziale del database. A seconda degli elementi che devono essere ricostruiti, il processo di esporta-

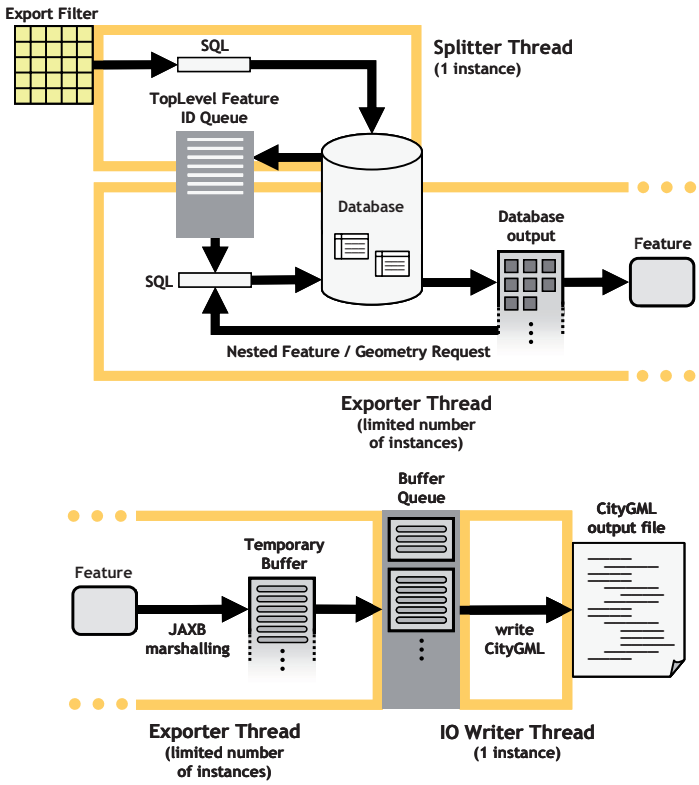

Figura 7.5: Dettaglio del workflow dai dati dal database filtrati per l'esportazione di una istanza di un documento CityGML

zione esegue interrogazioni più o meno complesse, che coinvolgono diverse tabelle per il reperimento dei dati degli elementi necessari. La funzionalità di esportazione forniscono il relativo codice per ogni tipo di funzione. In questa fase, la maggior parte della elaborazione dei dati viene effettuata dal server del database a causa di un uso efficiente di dichiarazioni di join SQL. Inoltre, se un singolo "worker thread" è in attesa di una risposta dal database, questo non blocca tutti gli altri thread all'interno dell'"export pool". Se tutti i dati sono stati ricevuti, essi sono mappati nei corrispondenti oggetti JAXB. Come in fase di importazione anche nella fase di esportazione deve essere tenuto traccia dei GMLID dei componenti già esportati, al fine di evitare i duplicati di oggetti in uscita e di utilizzare gli XLink invece. Gli oggetti JAXB sono trasformati(marshall) in eventi SAX , che sono memorizzati in un buffer temporaneo. Gli eventi SAX bufferizzati vengono piazzati nella coda del pool di "thread subsequent". Successivamente, gli "export thread" continuano a lavorare sulle successive feature. L'ultimo step del processo di esportazione, cioè, la scrittura di tutte le data feuture nell'istanza del documento CityGML. È svolta attraverso un I/O "writer thread". Questo thread prende gli eventi SAX nel bufferizzati dalla "work queue" e li traduce in elementi XML. Ancora una volta, questi processi basati su file sono disaccoppiati da tutte gli altri step della catena di processi.

Vi anche la possibilità mentre si estraggono i dati di poter trasformare le coordinate proiettandole su un sistema di riferimento diverso da quello utilizzato nella base dati. Questa funzionalità è però ancora in fase di sviluppo e non sempre va a buon fine. Per poter selezionare il sistema di riferimento si deve prima inserirlo nella preferenze sezione database come abbiamo gia visto, una volta inserito si potrà prima di esportare sceglierlo nel menu a tendina alla voce Reference System.

Esportazione d'istanze di documenti KML/COLLADA

Il processo per l'esportazione di istanze di documenti KML ricalca in modo fedele le fasi principali viste per l'esportazione di istanze di documenti CityGML. La semplificazione più importante è dovuta al fatto che non sarà necessario esportare tutte le top-feature ma è di interesse per KML ll sole feature building che rappresentano gli edifici che si possono rappresentare nel modello KML le altre feature non hanno un senso nell'ambiente KML. Un'altra cosa che lo distingue dal processo per citygml è che nella fase marshaling delle feature lo schema di riferimento non è lo schema di CityGML ma sarà a seconda delle modalità di esportazione scelta , tra lo schema di KML o lo schema COLLADA. Quest'ultimo viene usato nel caso si voglia inserire nelle istanze di documenti KML estratti oggetti COLLADA, in questo caso sarà necessario creare sia una istanza per il documento KML e solitamente per ogni edificio creare una istanza di documento COLLADA.

7.4 Interfaccia grafica

7.4.1 Connessione al database

Per la connessione al database PostgresSQL è necessario inserire le credenziali di accesso quali: username, password, indirizzo IP del server. porta di ascolto del server (di default postgreSQL usa la porta 5432), il nome del database e una descrizione per identificare la connessione. tutte queste informazioni sono inserite nel form Database. Come si vede nella Figura ref1 dal form database nella sezione "Connection details" è possibile creare nuove connessioni, salvare le connessioni, cancellare o copiare i dati relativi alle connessioni memorizzate; e naturalmente connettersi o disconnettersi al database. Quindi è possibile salvare connessioni a più database diversi contenenti diversi modelli CityGML. Le informazioni relative alla connessioni sono salvate nel file project.xml, così, come vedremo in seguito, anche tutte le altre informazioni inserite in tutti gli altri diversi form. In questo modo ogni volta che si avvia l'applicazione si caricano i dati relativi all'ultima sessione. L'applicazione comunica con il database attraverso l'interfaccia JDBC il cui funzionamento è stato esposto nei paragrafi 7.6. Nel form database è anche presente la sezione "Database operation" che si attiva solo dopo la connessione, qui è possibile estrarre dei dati dalla base dei dati al quale si è connessi. Si possono scegliere due operazioni cliccando sul radio botton e poi premendo il bottone execute, le operazioni sono le seguenti: generare un report e calcolare il massimo

boundig box. La prima operazione genera un report nella console che indica quanti elementi contenuti in ciascuna tabella presente nel database. L'operazione che calcola il massimo bounding box una volta eseguita restituisce il boundig box che contiene tutte le feature di alto livello scelte nel menu a tendina(top level feature) presenti nel database. É inoltre possibile scegliere anche con quale sistema di coordinate di riferimento si vuole esprimere il bounding box.

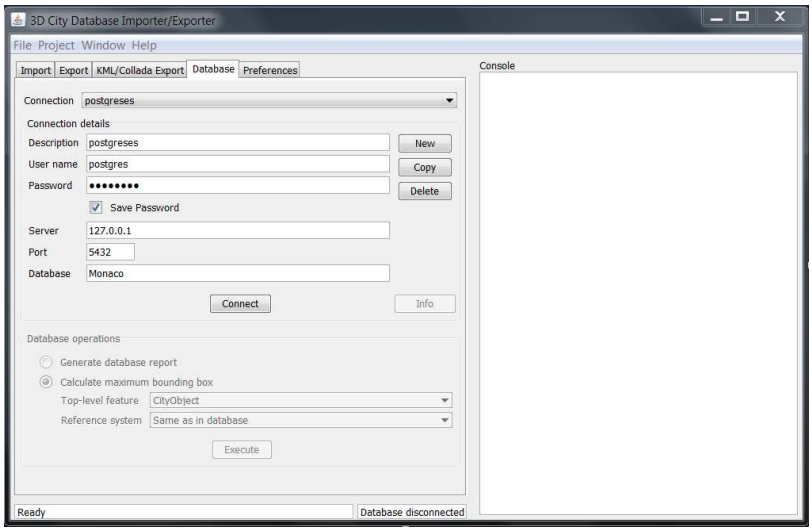

Figura 7.6: Form Database

7.4.2 Preferenze database

La scheda Preferences si suddivide in 5 diverse categorie: import, export, kml/collada exporter, database e general settings. Tutti i parametri modificati in questa scheda sono salvati nel file project.xml che si trova nella cartella di configurazione creata dall'applicazione, e sono caricati automaticamente durante l'avvio del programma. Con i pulsanti restore, default e save si possono, rispettivamente, ripristinare i valori precedenti alla modifica, resettare i valori di default o salvare i valori nel file XML. La sezione database si divide in due sottomenu Indexes e Reference System.

Indexes

In questa sezione si possono effettuare operazioni sugli indici del database, è divisa in due sezioni speculare una per gli indici spaziale e l'altra per gli indici comuni. In Entrambe le sezioni è possibile attivare e disattivare manualmente gli indici premendo il bottone active o disactive. E possibile inoltre attraverso un radio botton scegliere le seguenti tre opzioni:

Keep index status con questa opzione nel caso venga fatta una qualsiasi operazione gli indici no verranno modificati, quindi se attivi rimarranno attivi se disattivati rimarranno disattivati.

deactive before import and automatically reactive con questa opzione prima di ogni importazioni gli indici vengono disattivati, e alla fine del processo vengono riattivati

Deactive before import and keep deactivated con questa opzioni prima di ogni importazioni gli indici vengono disattivati e una volta finita l'operazione di importazione rimangono disattivati.

Reference System

In questa sezione si possono inserire le definizioni dei sistemi di riferimento supportati dal toolkit. Questi sistemi di riferimento inseriti sono utilizzati quando si vuole cambiare il sistema di riferimento rispetto a quello di default inserito nella tabella DATABASE_SRS. Vi sono due modalità per inserire le informazioni riguardanti i sistemi di riferimento o attraverso una form o caricando un file di testo opportunamente strutturato. I dati necessari per poter inserire il sistema di riferimento sono:

- SRID - è il sistema di riferimento utilizzato dalla base dati.
- gml:srsName - srsName rappresenta il nome del sistema di riferimento per i modelli CityGML
- descrizione - è una semplice descrizione

cliccando sul il bottone check si può controllare se SRID inserito è supportato dalla base dati.

7.4.3 Importazione file CityGML

La form d'importazione è suddivisa in tre parti: per la selezione di directory file, per definire operazioni sui filtri e per monitorare il flusso del programma. **selezione file** Esistono due modi per selezionare i file:

- attraverso la funzione Drag and Drop trascinando il file fino all'area che contiene la lista dei file da importare;
- premendo il tasto Browse, il quale apre una lista di Folder forniti dal sistema operativo. E si selezionano i file interessati presenti nel Folder.

Più di un file può essere selezionato. Tutti i file verranno importati. I file possono essere rimossi dalla lista, selezionando il corrispondente file della lista e cliccando sul bottone remove.

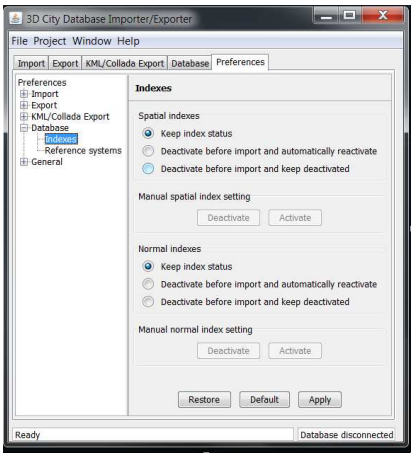

Figura 7.7: indici normali/spaziali

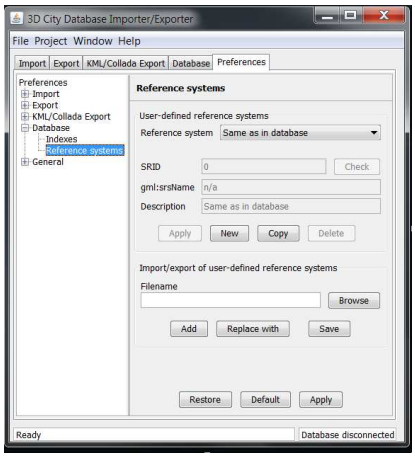

Figura 7.8: sistema di riferimento

Figura 7.9: Form preferenze database

Filtri d'importazione I radio button sono usati per selezionare gli oggetti che devono essere inclusi nel database. Ci sono due principali alternative che permettono il controllo dei dati in fase di importazione

GML-ID Filter Inserire in questo campo i GMLID degli oggetti che si vogliono importare. Multipli ID sono separati attraverso virgole. Solo gli oggetti indicati verranno importati dal file nel database.

Coplex Filter Questa caratteristica controlla l'importazione dei dati e consente multipli criteri di selezione per più tipi di filtri. Se più tipi di filtri sono selezionati essi sono combinati attraverso l'operazione AND. Essi sono attivati se sono selezionati nella checkbox. Se non sono selezionati i filtraggio non è applicato.

GML Name Filter Seleziona solo l'oggetti che ha il GML_NAME inserito e lo importa includendo tutte le sue subfeature. Solo un singolo GML Name può essere accettato.

CityObjectMembers/AppearanceMembers/FaetureMembers Se si prevede d'importare solo un sottoinsieme di oggetti, appearance members, o feature, può essere ristretta l'importazione a *n* feature di alto livello partendo dalla feature *m* seguito dagli *n* elementi presenti nel dataset.

BoundingBox Nell'importare oggetti in un'area d'interesse, per identificare il BoundingBox sono necessarie le coordinate in basso a sinistra e in alto a destra dell'area interessata. In accordo con le preferenze indicate nelle opzioni "BoundingBox ", gli oggetti importati sono parzialmente o completamente sovrapposti all'area d'interesse.É possibile inoltre selezionare anche quale sia il sistema di riferimento al quale appartengono le coordina-

te inserite, in modo tale che possa anche essere diverso da quello utilizzato all'interno del database

FeatureClasses Le feature delle classi pertinenti sono importate selezionando il corrispondente checkbox. Solo le classi di feature top-level. Se, ad esempio, scegliamo building, allora anche tutte le feature in esso contenute come buildingPart, WallSurface etc. sono importate.

Dopo aver selezionato i file da importare, i filtri e impostato i dati necessari per la connessione nel form database, premendo il bottone import si inizia l'importazione dei dati nel database.

Validazione XML

Nella form di importazione permette di validare le istanze dei documenti CityGML confrontandolo con lo schema XSD ufficiale di CityGML. Questa operazione può essere molto utile per individuare documenti di istanze di CityGML non conformi che potrebbero causare delle anomalie in fase di importazione. E raccomandato che prima di ogni importazione sia effettuata la validazione delle istanza di documenti CityGML che si vogliono importare. Il processo di validazione dopo aver selezionato il file parte se si preme il bottone *Just Validate*.

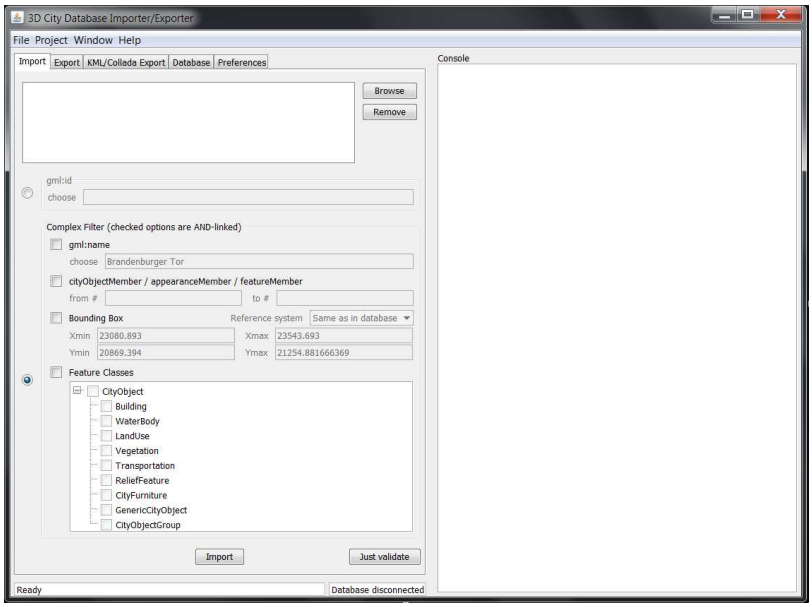

Figura 7.10: Form Importazione istanze di documenti CityGML

7.4.4 Preferenze Importazione

In questa sezione l'utente può impostare preferenze relative a:

- *ID Handling* - gestione degli identificativi
- *Appearance* - gestione delle appearance
- *XML validation* -validazione documenti XML
- *Resources* - gestione risorse

ID Handling

Identificatori univoci degli oggetti sono indispensabili, per evitare conflitti nei nomi in caso di aggiornamenti futuri. L'unicità può essere garantita generando automaticamente un identificatore universale univoco ("Universally Unique Identifier", UUID).

Come si può vedere nella Figura7.13, l'utente può decidere se usare gli UUID nel caso di ID omesso, in questo modo a tutti gli oggetti GML che non hanno un attributo gml:id viene assegnato un nuovo UUID. Oppure, può scegliere se rimpiazzare tutti gli ID con gli UUID ed eventualmente, memorizzare gli ID originali come referenze esterne, solo per i CityObject. Per gli UUID si può impostare anche il code space, quindi si può decidere se usare: il nome del file da importare, il path complemento del file da importare, oppure un descrittore definito dall'utente. In quest'ultimo caso si propone di default il valore UUID.

È preferibile rimpiazzare gli ID con UUID se il dataset da importare non contiene valori di gml:id globalmente univoci

Appearence

Oltre alla geometria degli oggetti, alla loro topologia e semantica, agli oggetti CityGML può essere assegnata una appearance. Attivando l'opzione corretta nel form riportato in Figura 7.13, si possono controllare l'importazione e la memorizzazione delle apparenze e delle texture.

In generale, si raccomanda di non usare elementi appartenenti alla classe TextureSurface, in quanto sono deprecati dalla versione 0.4.0. Invece di seguire questo metodo, si suggerisce di usare il concetto di Appearance di CityGML. Comunque, oggetti del tipo TextureSurface possono essere importati, saranno automaticamente trasformate nel nuovo modello di Appearance.

Ad ogni superficie può essere assegnata una o più apparenze, ognuna delle quali si riferisce ad un lato della superficie.

Le opzioni tra cui l'utente può scegliere sono:

- Importare appearance:
  - Importare appearance, importare file texture, (impostazione di default) nel caso in cui si scelga questa opzione, tutte le immagini delle texture riferite dal file CityGML, sono caricate nel database.
  - Importare le apparenze, e non i file delle texture.

  - Non importare le apparenze - in questo caso, le informazioni relative alle apparenze non sono importate nel database. La scelta di questa opzione influenza notevolmente le performance dell'applicazione in fase di importazione, perché il caching dei gml:id riferiti agli oggetti geometrici di tipo gml:LinearRing possono essere omessi.

- Conversione da TextureSurface ad Appearance:

  - salva nell'attributo theme - questo attributo è obbligatorio per la classe Appearance in CityGML dalla versione 0.4.0. "rgbTexture" è il valore di default che indica che le apparenze importate saranno delle texture.

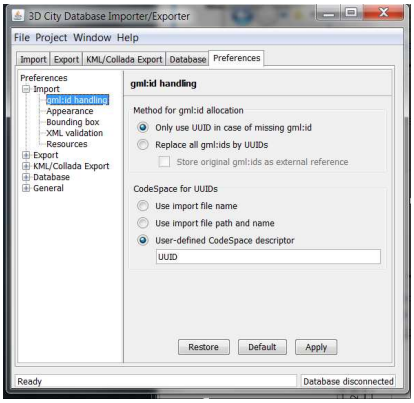

Figura 7.11: gml:id handling

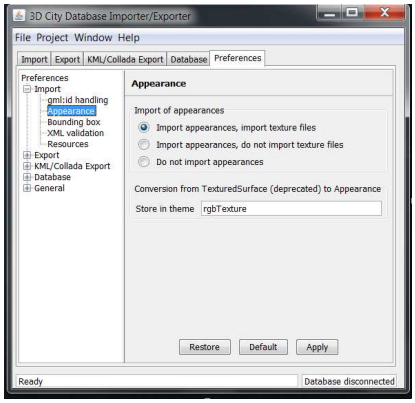

Figura 7.12: Appearance

Figura 7.13: le prime due sezioni del form preferenze import

XML Validation

L'applicazione permette di validare i documenti CityGML sulla base dello schema XML ufficiale di CityGML. Questo significa, che per essere importato un file deve essere valido rispetto allo schema di CityGML. File non validi possono causare un comportamento inaspettato del processo di importazione oppure una sua cessazione anomala. Per questo si raccomanda di importare solo file CityGML validi, questo migliora la robustezza del software.

Ogni documento che deve essere validato, è analizzato a pezzi invece di processare l'intero documento in una volta sola come in ogni altro programma per la validazione di file XML. Questo riduce il consumo della memoria centrale e rende possibile la validazione di documenti di dimensioni arbitrarie.

Si noti che la funzionalità di validazione è stata progettata ad-hoc per l'applicazione in oggetto. Infatti questa non risponde a tutti gli scopi della validazione XML. La validazione è realizzata solo per contenuti XML associati allo spazio dei nomi di

CityGML, quindi tutti i costrutti del documento che non rispondono a questo standard sono saltati e non valutati.

Il processo di validazione può essere avviato premendo il bottone Validate nella form di importazione, oppure essere eseguito in automatico per ogni file che verrà importato nel database abilitando la giusta opzione nella form mostrato in Figura 7.16. Da quest'ultimo form l'utente può scegliere di:

- Validare il file durante il processo di importazione.

  Se scegliamo questa opzione ogni file in input sarà validato. Se il processo di validazione rileva degli oggetti non congrui allo schema CityGML, allora questi non saranno importati.

- Schema XML da usare per la validazione.

  – Usare l'attributo xsi:schemaLocation per l'elemento radice. Attivando questa opzione l'attributo xsi:schemaLocation nell'elemento all'inizio del documento è cercato per lo schema XML da usare per la validazione. Quest'opzione, però non è consigliata perché non è garantito che il file di input segua lo schema ufficiale di CityGML. Tuttavia, nel caso in cui il documento in esame contenga codice XML proveniente da CityGML Application Domain Extension (ADE), allora questo è l'unico modo per validare correttamente il file fin tanto che i contenuti ADE sono specificati in uno proprio schema XML.

  – Usare una copia locale degli schemi:

    * CityGML v1.0.0 Base Profile
    * CityGML v0.4.0.

- Altre opzioni di validazione.

  – Riporta solo un errore per gli oggetti di livello superiore. Questa riduce altamente il numero di messaggi di errori sulla console.

Resources

L'architettura dell'intero programma è basata sul multithreading, ovvero sull'esecuzione concorrente di più task. Generalmente, ogni task dei processi di importazione/esportazione dei dati CityGML è eseguito da thread separati.

Inoltre, l'applicazione usa delle strategie per trattare documenti CityGML di dimensioni arbitrarie e risolvere gli XLink. Finché sono possibili le referenze all'indietro, la risoluzione degli XLink necessita della presenza dell'intero documento in memoria. Naturalmente, questo rappresenta un grande limite per quel che riguarda la massima dimensione del documento di input. Per questo motivo, il processo di importazione segue un strategia a due fasi, in cui in un primo momento gli oggetti sono scritti nel

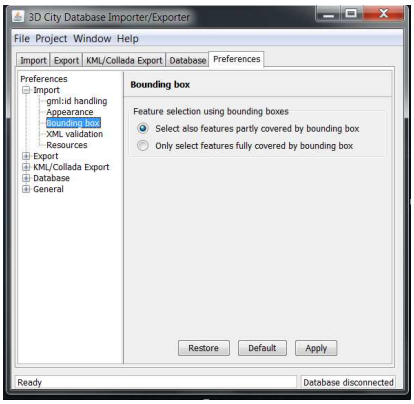

Figura 7.14: Bounding Box

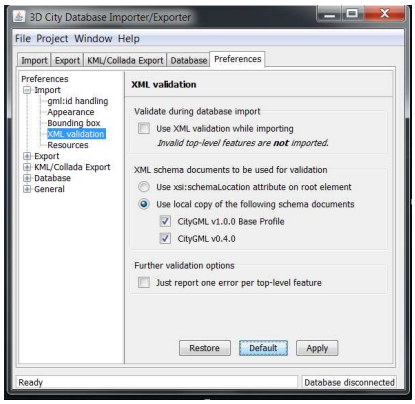

Figura 7.15: XML validation

Figura 7.16: le terza e la quarta sezione del form preferenze import

database trascurando le referenze agli oggetti remoti. Comunque, se un elemento contiene uno XLink, ogni informazione relativa allo XLink è scritta in tabelle temporanee nel database.

Inoltre, durante l'analisi del documento il processo di importazione tiene traccia di ogni gml:id incontrato così come del nome della tabella, della chiave primaria del corrispondente oggetto nel database. Questo è importante per memorizzare queste informazioni perché a priori non si può prevedere se un gml:id è riferito da uno XLink oppure no. In modo tale da assicurare un accesso veloce, le informazioni sono copiate temporaneamente in memoria. Se la dimensione massima della cache è raggiunta, allora la cache è svuotata riversandone il contenuto in tabelle temporanee del database, così da prevenire un overflow della memoria. In un secondo momento, le tabelle temporanee contenenti informazioni contestuali sugli XLink sono ricontrollate. Affinché l'intero documento CityGML sia processato, le referenze valide devono essere risolte e trattate di conseguenza. Con l'aiuto dei gml:id presenti nella cache, gli oggetti riferiti possono essere ritrovati velocemente all'interno del database. Il comportamento della cache, per esempio la sua dimensione, può essere influenzato attraverso il form riportato nella Figura 7.17.

Infine, per ottimizzare i tempi di risposta del database, più operazioni sono incapsulate in un sola richiesta che sarà inviata al database (batch processing). Il numero di istruzioni SQL presenti in un gruppo può essere stabilito dall'utente attraverso il form relativo alle impostazioni delle risorse per l'importazione Figura 7.17. Queste impostazioni riguardano:

- Esecuzione multithreaded:

  Numero minimo/massimo di thread. Questi valori dipendono dalla configurazione hardware del computer su cui è eseguito il programma. I valori di default corrispondono al numero di core della CPU moltiplicati per due.

Prima di iniziare il processo di importazione il numero minimo di thread è creato. Thread aggiuntivi al numero massimo di thread sono creati solo se necessario. Si noti che un numero maggiore di thread non fornisce, necessariamente, performance migliori. Anzi, un numero troppo elevato di thread attivi può causare degli svantaggi. Per esempio, nella JVM, la creazione di troppi thread può causare un errore di tipo "out of memory" dovuto al consumo eccessivo di memoria.

- Batch processing:

  Tutte le operazioni del database di un gruppo (batch) sono bufferizzate in memoria, prima di inviare il gruppo di richieste al database. Quindi, l'applicazione può andare in crash se il campo batch_size ha un valore troppo. alto.

  - *Top level features* - valore di default 20.
  - *Gml:id cache entries* - valore di default 10.000
  - *Temporary information* - valore di default 10.000. Usato per memorizzare le informazioni relative agli XLink.

- Gml:id cache. Usato per risolvere gli XLink.

  - Geometry/Features:
    - *Entries* - numero massimo di gml:id da tenere in memoria. Se questo numero è troppo alto il programma può avere problemi di memoria. Il valore di default è 200.000.
    - *Page factor %* - parametro correlato al numero di gml:id che si possono memorizzare nella cache. Definisce la percentuale delle voci della cache che sono inviate al database se si raggiunge la dimensione massima della cache. Il valore di default è 85%.
    - *Table partitions* - rappresenta il numero di tabelle temporanee usate per il trasferimento dei gml:id nel database. Il valore di default è 10.

7.4.5 Esportazione file CityGML

Anche la form d'esportazione come la form d'importazione è suddivisa in tre parti: per la selezione di directory file, per definire operazioni sui filtri e per monitorare il flusso del programma.

Selezione file La selezione dei file di output è simile a quella di importazione. E necessario premere il bottone "browse" e poi selezionare il folder e digitare il nome del file. Infine confermare la richiesta.

Filtri d'esportazione I radio button sono usati per selezionare gli oggetti che devono essere inclusi nel database. Ci sono due principali alternative che permettono il controllo dei dati in fase di esportazione:

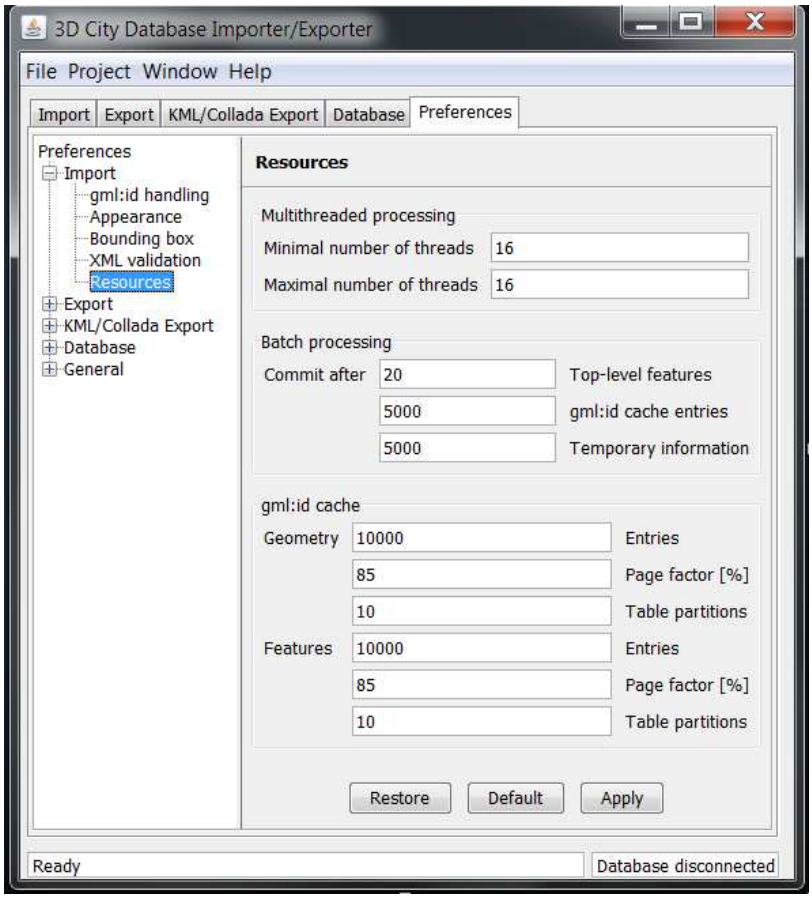

Figura 7.17: l'ultima sezione delle preferenze di importazione

GML-ID Filter Inserire in questo campo i GMLID degli oggetti che si vogliono esportare. Multipli ID sono separati attraverso virgole. Se il GML-ID si riferisce ad un gruppo di oggetti, tutti i membri del gruppo vengono esportati.

Coplex Filter Questa caratteristica controlla l'importazione dei dati e consente multipli criteri di selezione per più tipi di filtri. Se più tipi di filtri sono selezionati essi sono combinati attraverso l'operazione AND. Essi sono attivati se sono selezionati nella checkbox. Se non sono selezionati il filtraggio non è applicato.

GML Name Filter seleziona gli oggetti che hanno il GML Name inserito, l'applicazione invia al database una interrogazione contenete queste informazioni, che estrae tutti gli oggetti, aggregati inclusi, corrispondenti. E permesso inserire un solo valore di GML Name.

CityObjectMembers/AppearanceMembers/FaetureMembers Se si prevede d'esportare solo un sottoinsieme di oggetti, appearance members, o feature, può essere ristretta l'esportazione a *n* feature di alto livello partendo

dalla feature ID pari a *m* seguito dagli *n* elementi presenti nella tabella CITYOBJECT.

BoundingBox Per esportare oggetti in un'area d'interesse, per identificare il BoundingBox sono necessarie le coordinate in basso a sinistra e in alto a destra dell'area interessata. In accordo con le preferenze indicate nelle opzioni "Bounding Box ", gli oggetti esportati sono parzialmente o completamente sovrapposti all'area d'interesse. É possibile inoltre selezionare anche quale sia il sistema di riferimento al quale appartengono le coordinate inserite, in modo tale che possa anche essere diverso da quello utilizzato all'interno del database.

FeatureClasses Le feature delle classi pertinenti sono esportate selezionando il corrispondente checkbox. Solo le classi di feature top-level selezionate vengono esportate. Se, ad esempio, scegliamo building, allora anche tutte le feature in esso contenute come buildingPart, WallSurface etc. saranno esportate.

Dopo aver selezionato i file da esportare, settato i filtri e impostato i dati necessari per la connessione nel form database, premendo il bottone export si inizia l'esportazione dei dati.

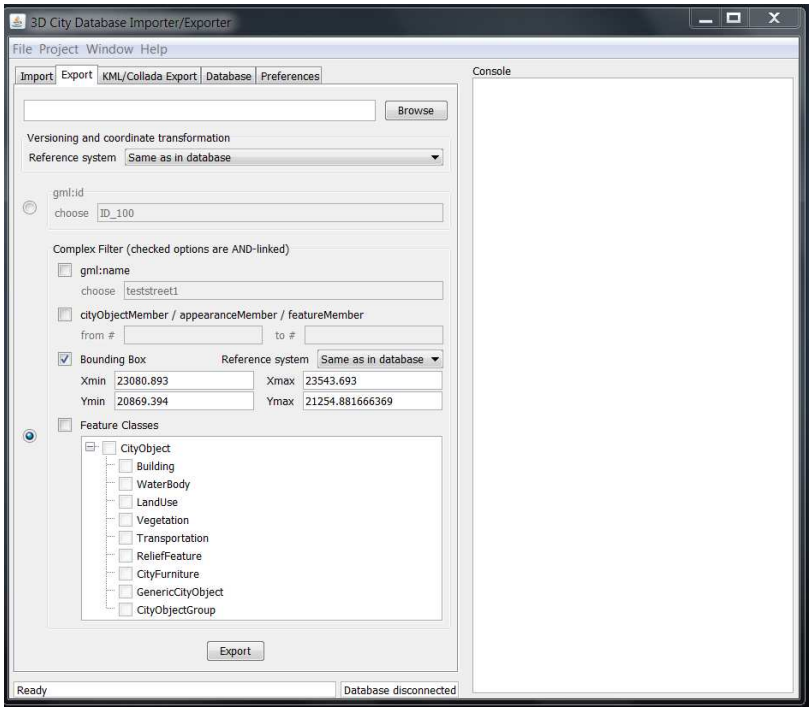

Figura 7.18: Form Esportazione CityGML

7.4.6 Preferenze Esportazione

La terza parte della scheda relativa alle preferenze riguarda le impostazioni per l'esportazione dei dati CityGML dal database. Queste, a loro volta, si suddividono in cinque aree:

- CityGML module.
- Appearance.
- Bounding Box.
- Xlink
- ResourcesAS

cityGML module

Il tool di esportazione permette di esportare dati in formato CityGML versione 0.4.0 o 1.0.0. In questa sezione l'utente può scegliere per ogni classe di CityGML (LandUse, Building, Vegetation, ...) quale versione usare, se la versione 0.4.0 o la 1.0.0. Nel form rappresentato in Figura 7.21 si può vedere come, l'utente può scegliere la versione di CityGML da usare, selezionandola dal menù a tendina associato ad ogni classe.

Appearance

Come per la procedura di importazione, anche in fase di esportazione, si possono gestire, in modo simile, le texture e le appearance.

L'utente può scegliere tra tre alternative:

Esportare le apparenze:

- opzioni per esportare le appearance:
  - *Esporta le appearance e salva i file delle texture* - impostazione di default. É anche possibile decidere se sovrascrivere i file esistenti oppure no, scegliendo la giusta opzione.
  - *Esporta le appearance e non salvare i file delle texture*
  - *Non esportare le appearance*.

Inoltre l'utente può impostare la cartella in cui esportare i file delle texture. Le opzioni fra cui può scegliere sono:

- Directory assoluta ovvero fornire l'indirizzo completo della cartella di destinazione.
- Sottocartella della cartella di esportazione, basta inserire il nome della cartella che verrà creata nella cartella di esportazione.

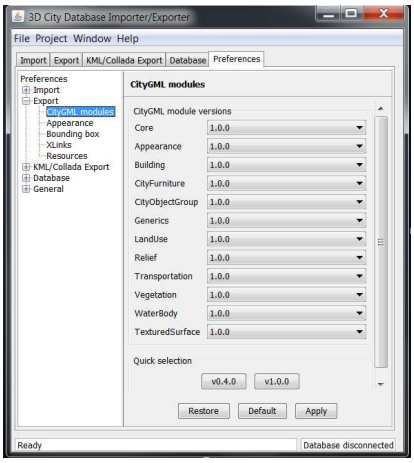

Figura 7.19: CityGMl modules

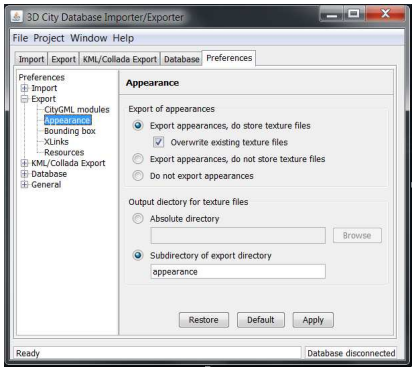

Figura 7.20: Appearance

Figura 7.21: le prime due sezione del form preferenze esportazione

Bounding Box

Da qui l'utente può scegliere se esportare gli oggetti completamente contenuti in una are di interesse limitata dal bounding box, oppure se esportare anche gli oggetti che intersecano soltanto l'area di interesse. Queste impostazioni sono usate quando si attiva il filtro per l'esportazione nella scheda dedicata, altrimenti sono ininfluenti, comunque ne parleremo in seguito. Inoltre si può settare l'operazione di tiling che divide l'area di interesse in piccole porzioni e per ognuna di esse crea un documento del modello Citygml che rappresenta la parte sezionata. Ciò permette di avere in esportazione file di dimensioni ridotte. Si può settare il numero di righe e colonne con le quali si dividera l'area di interesse attraverso i campi rows e columns. É è possibile selezionare il nome della sottodirectory che conterrà i file prodotti in esportazione con l'operazione di tiling. Inoltre è possibile anche scilegliere il suffiso da applicare alle cartelle e ai file esportati. Nella sezione Further tilling options e possibile inserire l'informazione di tiling anche come attributo generico all'interno nel modello CityGML estratto, si possono selezionare nome e valore che verranno inseriti.

Xlinks

La sezione Xlink si divide in due parti:

Multiple export of feature elements in questa sezione è possibile gestire Gli Xlink prodotti in esportazione per quando riguardano le feature. Si possono sciegliere due modalità:

- i riferimenti agli Xlink vengo associati ad un elemento feature esistente.
- i riferimenti agli xlinkvengono riferiti ad una copia con un nuovo identificativo.

Multiple export of geometry elements in questa sezione è possibile gestire Gli Xlink prodotti in esportazione per quando riguardano le geometrie. Quanto visto per le feature vale anche per le geometrie.

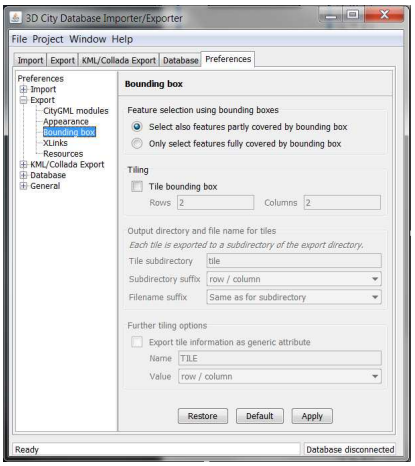

Figura 7.22: Bounding Box

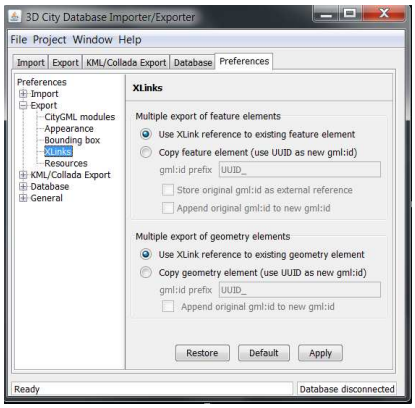

Figura 7.23: XML validation

Figura 7.24: le terza e la quarta sezione del form preferenze export

Resources

La procedura di esportazione, così come quella di importazione, è basata sul multithreading, così da migliorare le prestazioni dell'applicazione. Le impostazioni delle risorse permettono di controllare, gestire, il numero di thread concorrenti usati durante la fase di esportazione

Per ricostruire gli XLink il programma deve tenere traccia di tutti i gml:id degli elementi esportati. Prima di esportare un oggetto si controlla se un oggetto con lo stesso gml:id non sia già stato analizzato. Per questo motivo, i gml:id sono memorizzati nella cache così da velocizzare queste operazioni. Come abbiamo già visto per l'importazione, quando la cache raggiunge la sua dimensione massima, questa è svuotata e il suo contenuto è riversato in tabelle temporanee del database.

I parametri modificabili dall'utente riguardano:

- Esecuzione multithreaded:
  - Numero minimo/massimo di thread. Questi valori dipendono dalla configurazione hardware del computer su cui è eseguito il programma. I valori di default sono il numero pari al numero di core della CPU moltiplicati per due. Prima di iniziare il processo di esportazione il numero minimo di thread è creato. Thread eccedenti il numero massimo sono creati solo

se necessario. Si noti che un numero maggiore di thread non fornisce, necessariamente, performance migliori. Anzi, un numero troppo elevato di thread attivi può causare degli svantaggi. Per esempio, nella JVM, la creazione di troppi thread può causare un errore di tipo “out of memory” dovuto al consumo eccessivo di memoria.

- Gml:id cache. Usato per risolvere gli XLink per le due sezioni Geometry e Features:
  - *Entries* - numero massimo di gml:id da tenere in memoria. Se questo numero è troppo alto il programma può vere problemi di memoria. Il valore di default è 200.000.
  - *Page factor %* - parametro correlato al numero di gml:id che si possono memorizzare nella cache. Definisce la percentuale delle voci della cache che sono inviate al database se si raggiunge la dimensione massima della cache. Il valore di default è 85%.
  - *Table partitions* - rappresenta il numero di tabelle temporanee usate per il trasferimento dei gml:id. Il valore di default è 10.

7.4.7 Esportazione file KML/COLLADA

La form di esportazione di modelli KML/COLLADA e formata da cinque sezioni: per la selezione di directory e file, per definire operazioni sui filtri, per selezionare il livello di dettaglio dal quale poter esportare, per selezionare lo stile di visualizzazione e una console per monitorare il flusso del programma.

Selezione file

La selezione dei file è la stessa utilizzata nella form exporter.

Export Contents

I radio button sono usati per selezionare gli oggetti che devono essere inclusi nel database. Ci sono due principali alternative che permettono il controllo dei dati in fase di esportazione:

Single building Inserire in questo campo i GMLID del building(Edificio) che si vuol esportare. Multipli ID sono separati attraverso virgole. Se il GML-ID si riferisce ad un gruppo di oggetti, tutti i membri del gruppo vengono esportati.

Bounding Box Per esportare oggetti in un’area d’interesse, per identificare il BoundingBox sono necessarie le coordinate in basso a sinistra e in alto a destra dell’area interessata. In accordo con le preferenze indicate nelle opzioni “Bounding

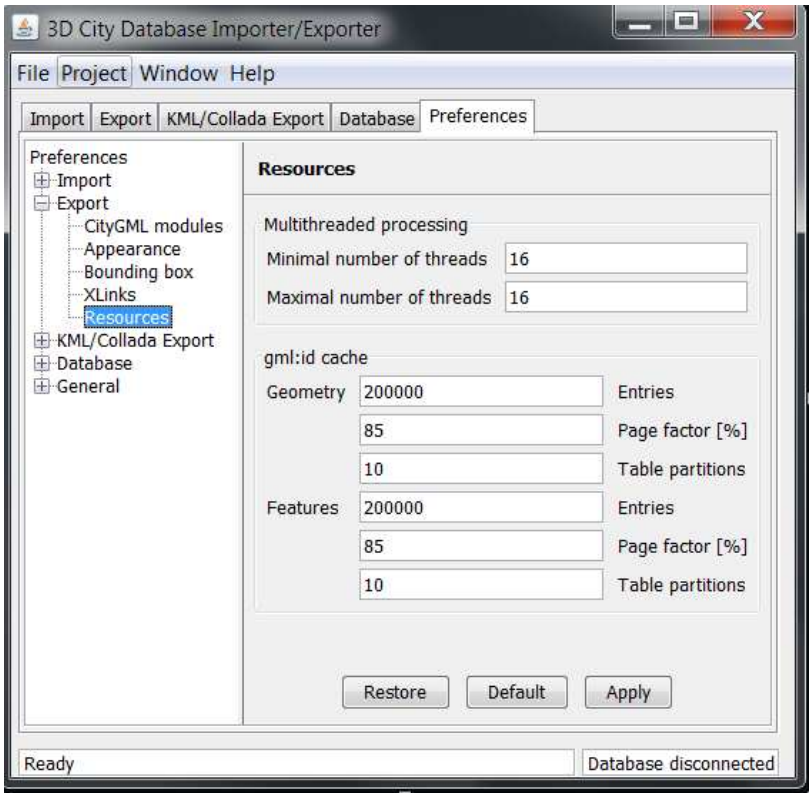

Figura 7.25: l'ultima sezione delle preferenze di esportazione il form resource

Box ", gli oggetti esportati sono parzialmente o completamente sovrapposti all'area d'interesse. É possibile inoltre selezionare anche quale sia il sistema di riferimento al quale appartengono le coordinate inserite, in modo tale che possa anche essere diverso da quello utilizzato all'interno del database. Se attivato il bounding box è possibile scegliere se effettuare la funzione di tiling che divide l'area di interesse in tante sottoaree, dove a ciascuna sottoarea verrà generato in uscita un file che rappresenta l'istanza del documento XML che rappresenta la sottoarea stessa. il tiling può essere:

- *no-tiling* disattivato.
- *automatic* il tiling è attivato è il numero di sottoaree create sono generate automaticamente in base alla dimensione dell'area di interesse indicata nel bounding box.
- *manual* viene attivato e l'area di interesse vie divisa in righe e colonne il cui numero è indicato dall'utenete nei capi rows e collumns.

Export for level of detail

In questa sezione decidiamo a quale livello di dettaglio andare a estrarre i dati contenuti nel database. Ad esempio se impostiamo nel menu a tendina il livello di dettaglio lod_2 allora quando andremo a esportare i dati verranno selezionate solo le geometrie che appartengono a questo livello di dettaglio e verranno scartate tutte le altre appartenenti a livello di dettaglio diversi.

Display as

In questa sezione si seleziona quale modalità di visualizzazione si vuol dare all'edificio una volta estratto. ci sono quattro modalità di visualizzazione:

Footprint: gli edifici sono rappresentati attraverso la loro proiezione sulla superficie della terra. Questa modalità si può applicare a tutti i livelli di dettaglio LOD visti in precedenza.

Extruded: gli edifici sono rappresentati come un modello a blocco, realizzato attraverso l'estrusione della footprint di un valore pari alla altezza dell'edificio (Altezza è un attributo tematico di CityGML). Si applica a tutti livelli di dettaglio tranne il LOD 0.

Geometry: mostra i dettagli geometrici del suolo, e delle superficie di pareti e tetti di edifici, e aggiunge informazioni attraverso la possibilità di applicare delle appearance. Ad esempio i tetti sono colorati di rosso mentre le pareti sono colorate di grigio. Questo modalità di visualizzazione richiede di avere a disposizione delle informazioni semantiche a disposizione nei dati presenti nella base di dati. In questa modalità le appearance che possiamo applicare sono limitate alla sola colorazione delle superficie, non si possono utilizzare le texture perché il modello KML di base non ne permette l'utilizzo.Si applica a tutti livelli di dettaglio tranne il LOD 0

COLLADA: Per ovviare al problema di non poter utilizzare le texture in questo livello di dettaglio si integra nel modello KML il modello COLLADA che da la possibilità di includere le texture. Quindi questo modalità di visualizzazione si comporta come il modello geometry e in più visualizza se presente le texture. Questo è anche l'unico modello nel quale non si usa solamente il formato KML. Si applica ai livelli di dettaglio LOD 2,3,4.

Esportare

Una volata configurato in maniera opportuna tutte le varie sezione cliccando sul pulsante export si da inizio al processo di esportazione.

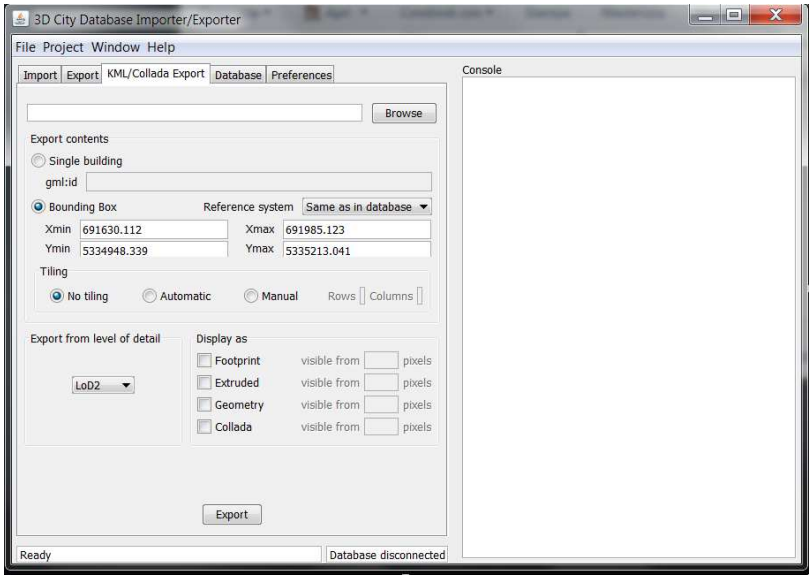

Figura 7.26: Form esportazione istanze di documenti KML/COLLADA

7.4.8 Preferenze KML/COLLADA

La sezione preferenze KML/COLLODA si suddivide in tre parti:

- Redering
- Balloon
- Altidude/Terrain

Rendering

la sezione rendering da la possibilità di attivare o disattivare funzionalità per la visualizzazione degli edifici esportati con il modello KML. si divide a sua volta in quattro sezioni:

kmz/tile borders prima sezione è presente un check box nella quale si può selezionare se i file generati con l'esportazione siano compressi con il formato kmz o no. E se il bordo dell'area definita dal tile sia visualizzato. Infine si può impostare la lunghezza in metri necessaria per poter effettuare l'operazione di tiling in automatico.

Footprint and Extruded Dispaly in questa sezione si possono settare i parametri per la modalità di visualizzazione footprint e extruded. Si può definire il colore utilizzato per il riempimento delle superfici e il colore associato alle linee. Attraverso il checkbox si può anche aggiungere oltre alla normale visualizzazione anche la possibilità di evidenziare l'edificio se ci si passa sopra con il mouse si può come

in precedenza scegliere il colore di riempimento delle superficie e delle linee che vengono evidenziate.

Geometry display option in questa sezione si possono settare i paramatri per la modalità di visualizzazione geometry. Come in precedenza si può definire il colore di riempimento delle superficie appartenenti alla categoria o pareti o tetto, e il colore delle linee di contorno delle superficie. Anche in questo caso è presente il check box per poter dare la possibilità di visualizzare l'edificio nel caso ci si passi sopra con il cursore del mouse.

Collada display option in questa sezione si possono settare i parametri per la modalità di visualizzazione del modello COLLADA.

- *Appearance/Theme* se presenti nel database theme diversi si può selezionare nel menù a tendina quello che si desidera esportare.
- *Generare texture atlases* se viene selezionata questa opzione nel caso vengano create delle texture queste vengono accorpate in una sola immagine utilizzando l'algoritmo che si può scegliere nel menu a tendina.
- *Hightlight when on Mouseover* nel caso venga selezionata questa opzione se si passa sopra con il mouse agli edifici si evidenzia una geometria che rappresenta l'edificio oltre a quella rappresentata con COLLADA come avviene anche per le altre modalità di visualizzazione.

Balloon

Alla visualizzazione geometrica è possibile aggiungere dei *balloon* riempiti dinamicamente con informazioni. In dettaglio ogni placemark può contenere una descrizione che può essere mostrata in un "fumetto" quando viene cliccato l'oggetto. Il contenuto può essere dinamicamente riempito nel momento dell'esportazione con specifiche informazioni. Per poter realizzare tutto ciò è necessario creare un template HTML con delle semplici istruzioni speciali come nell'esempio che segue:
BallonSource.html

```
[...]

    <table width=400>
    <tr><td><b>Address:</b></td></tr>
        <tr><td>
            <3DCityDB>ADDRESS/[FIRST]STREET</3DCityDB>
            <3DCityDB>ADDRESS/[FIRST]HOUSE_NUMBER</3DCityDB>
        </td></tr>
    <tr><td> </td></tr>
    <tr><td><b>Envelope:</b>
      <3DCityDB>CITYOBJECT/ENVELOPE</3DCityDB>
[...]
```

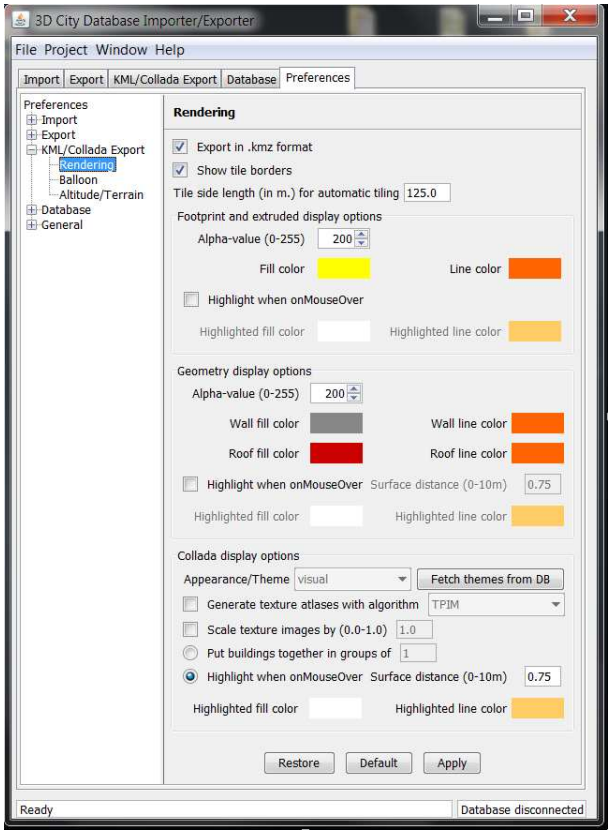

Figura 7.27: form preferenza rendering per l'esportazione KML/COLLADA

```
    </td></tr>
        <tr><td> </td></tr>
    <tr><td><b>Appearance:</b>
      <3DCityDB>APPEARANCE/[COUNT]ID</3DCityDB>
    </td></tr>
    <tr><td><b>Measure Height:</b>
      <3DCityDB>BUILDING/[FIRST]MEASURED_HEIGHT</3DCityDB>
    </td></tr>
    <tr><td><b>External referance name:</b>
      <3DCityDB>EXTERNAL_REFERENCE/[FIRST]NAME</3DCityDB>
    </td></tr>
  </table>
[...]
```

I tag 3DCityDB è un tag speciale utilizzato per identificare le istruzioni da effettuare in fase di esportazione sulle tabelle della base di dati. La struttura della richiesta è la seguente:
<3DCityDB>NOMETABELLA/[FIRST/LAST/COUNT]NOMECOLONNA</3DCityDB>
per quanto riguarda la sezione preferenze balloon dà la possibilità di attivare e disatti-

vare la visualizzazione dei ballon vediamo in dettaglio quali sono i possibili settaggi:

Placemark must include è una check box che se spuntata dà la possibilità di generare i balloon.

Ballon content source in questa sezione è possibile scegliere il baloon. La prima opzione e di reperire le informazioni dalla tabella cityobject_genericattrib. La seconda opzione è quella di caricare un template definito come mostrato in precedenza.Con questa seconda opzione vi è la possibilità di utilizzare il template solo nel caso non vi siano generic_attribute

Export ballon contenets into separate file for each object se selezionata crea dei file esterni al file kml per contenere le informazioni per riempire i balloon.

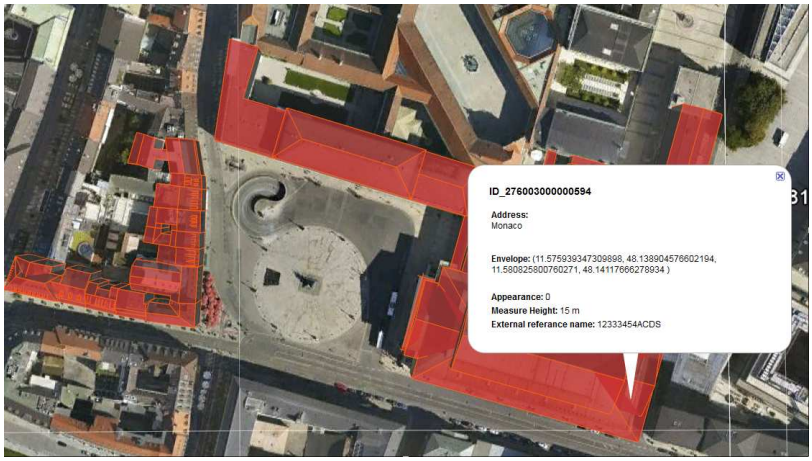

Figura 7.28: Esempio visualizzazione Balloon

Altitude/Terrain

I valori delle coordinate dell'altezza nel sistema di riferimento 3DCityDB potrebbero non corrispondere con quelle del DTM (Digital Terrain Model) di Google Earth (CRS WGS84/Geoid92). Il risultato di questa discrepanza potrebbe essere quello di avere edifici sospesi per aria o sprofondati sotto terra come mostrato in Figura 7.31. Per risolvere questo problema: si deve definire l'altitudine assoluta(absolute) con il tag <altidudeMode>, richiamare la API GoogleElevation per ottenere il punto della pianta dell'edificio con il valore più basso dell'altezza del DTM, sottrarre al coordinata z del punto il valore elevetion calcolato di questo punto in modo tale da ottenere così un valore di z-offset. Applico poi il valore z-offset a tutte le coordinate z dell'edificio.

La sezione di preferenze Altitude/Elevation Figura 7.34si divide in due sezioni:

Altitude mode: seleziona la modalità di altitudine da applicare ai documenti del modello KML. Si possono scegliere due modalità:

Figura 7.29: Corretto

Figura 7.30: Errato

Figura 7.31: errore nella posizionamento dell'edificio che sprofonda nel suolo

- Absolute: la misura dell'altezza è riferita sul livello del mare
- Relative: la misura dell'altezza e riferita al livello del suolo.

Altitude offset: seleziona la modalità di calcolo dello z-offset vi sono tre opzioni:

- no offset: lo z-offfset è posto pari a zero.
- Constant: lo z-offset viene posto uguale al valore costante inserito in metri
- Use Generic attribute: l'offset può essere calcolato attraverso Google Elevation.

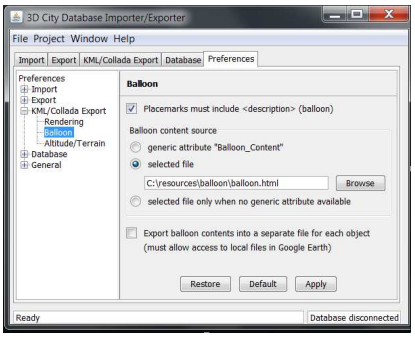

Figura 7.32: Balloon

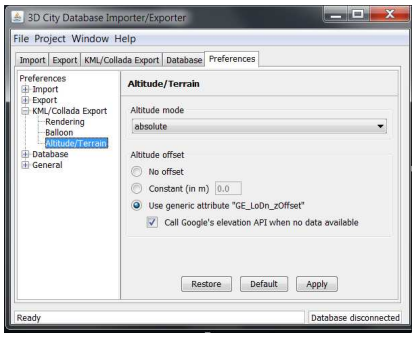

Figura 7.33: Altitude/Terrain

Figura 7.34: seconda e terza sezione del form preferenze KML/COLLADA

7.4.9 Preferenze Generali

La sezione preferenze General. A sua volta si suddivide in tre parti:

Logging I messaggi di LOG sono usati per registrare informazioni relative ad eventi quali attività o errori, per esempio nelle fasi di importazione o esportazione.

Questi messaggi includono informazioni relative al tempo in cui si è verificato l'evento, ma anche relative alla gravità di questo. I messaggi di LOG sono sempre visualizzati sulla console, e possono essere salvati in un file di LOG nel computer su cui è in esecuzione l'applicazione. Si possono distinguere quattro diversi livelli di LOG, dal più grave al meno grave:

- ERROR - quando si verifica un errore nel programma, solitamente un'eccezione.
- WARN - quando si individua una condizione anomala e il programma tenta di comunicare con essa.
- INFO - informazioni di carattere generale, ad esempio connessione instaurata oppure fine del processo di import.
- DEBUG - messaggi aggiuntivi che riportano lo stato interno del programma.

L'utente può scegliere, usando il menù a tendina LOG level, il livello minimo per i messaggi di LOG. Può, inoltre, scegliere se scrivere i messaggi di LOG in un file e il livello minimo di gravità per cui scrivere i LOG su file

Import export path Da qui l'utente può scegliere se tenere come path predefinito l'ultimo path usato per l'importazione o per l'esportazione, o se usare sempre la stessa cartella indicata. Si possono anche specificare cartelle diverse per l'importazione e l'esportazione.

Language selection Da qui è possibile scegliere la lingua utilizzata nei menu (lingue ad ora disponibili inglese e tedesco).

File di configurazione

Tutte le configurazioni vengono salvate nel file project.xml nel momento di chiusura del toolkit. Altre a questo salvataggio è possibile cliccando sulla barra del menu alla voce Project si apre un menu che riguarda la gestione di questo indispensabile file. Di seguito viene illustrato in dettaglio questo menu:

- **Open Project** dà la possibilità di caricare i settaggi presenti in un file project.xml, dando la possibilità di cercare il file in folder diversi da quello predefinito.
- **Save Projet** dà la possibilità di memorizzare i settaggi nel file project.xml di default, in quel momento configurati nel toolkit
- **Save project as** dà la possibilità di memorizzare i settaggi nel file project.xml indicato anche diverso da quello di default, in quel momento configurati nel toolkit.
- **Restore default Setting** ripristina le configurazioni di default a tutto il toolkit.

- **Save project XSD as** salva in un folder indicato un file XSD che contiene lo schema per il file project.xml

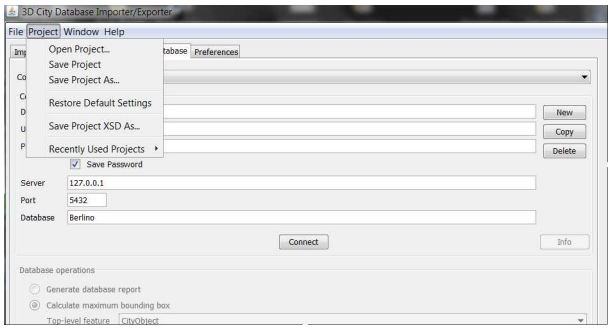

Figura 7.35: Menù project

7.5 Interfaccia a linea di comando

É possibile eseguire il toolkit anche da linea di comando basta aprire un shell e digitare i seguenti comandi:
java -jar 3dcitydb_postgis.jar $[-options]$
il file jar si trova nella cartella script del progetto. Per poter visualizzare un help si digita il seguente comando:
java -jar 3dcitydb_postgis.jar -help

Nella Figura 7.36 è mostrato il menu di help del programma.

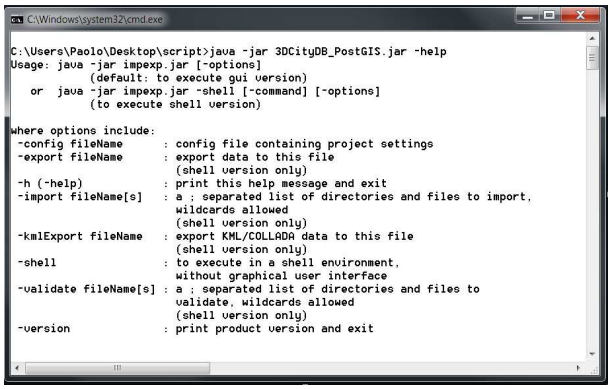

Figura 7.36: Menù Help console

Essenziale che il file di configurazione (project.xml) sia configurato in maniera opportuna. In questo file sono innanzitutto memorizzate informazioni concernenti la

connessione al database e tutti i settaggi visti in precedenza che si possono settare nell'interfaccia grafica. Lo stesso file è utilizzato per memorizzare le configurazioni scelte attraverso l'interfaccia grafica. Per questo può risultare utile utilizzare proprio l'interfaccia grafica per configurare questo file in maniera più agevole per poi riutilizzarlo anche nel caso si usi l'interfaccia a linea di comando.

Capitolo 8

Conclusioni

Il lavoro svolto in questa tesi ha portato alla realizzazione di una base dati centralizzata che memorizza in maniera efficiente e completa le istanze di documenti CityGML, e ad un toolkit di interfaccia tra l'utente e la base dati che permette importazione e l'esportazione dei modelli CityGML. Il tutto è sviluppato in un ambiente opensource. In dettaglio la base dati è stata sviluppata utilizzando ORDBMS PostgreSQL ver(8.4) , al quale è stato integrata l'estensione spaziale PostGIS (ver.1.5.2) Al DBMS è stato applicato lo schema relazionale realizzato ad hoc per poter memorizzare in maniera efficiente e completa istanze di documenti CityGML. Il toolkit è completamente scritto in JAVA; importa e esporta modelli CityGML da e verso la base dati per tutti il livelli di dettaglio e per tutte le feuture, sia in fase di importazione che in fase di esportazione è possibile applicare dei filtri. Al toolkit inoltre per promuovere una maggiore fruibilità dei dati contenuti nella base di dati è stato aggiunta la possibilità di esportare anche istanze di modelli KML attraverso una opportuna mappatura dei dati presenti nella base dati in formato CityGML.

8.1 Sviluppi futuri

Il progetto ha raggiunto un discreto livello di maturità ma deve essere estensivamente testato ed ha ampi margini di miglioramento per quanto riguarda prestazioni e stabilità. Si possono incrementare alcune funzionalità, in particolare sarebbe opportuno espandere i filtri messi a disposizione per un maggiore controllo delle fasi di importazione e esportazione.
Anche a livello di stili si possono espandere le funzionalità di vestizione dinamica, colorando gli oggetti sulla base di altri attributi decisi dall'utente. Altro punto che si è già cominciato a sviluppare ma non dà ancora i risultati voluti, è quello di poter, in fase di estrazione, effettuare una conversione del sistema di riferimento.
Infine, per completare il lavoro già intrapreso con l'esportazione di modelli KML, si potrebbe continuare su questa via ed estrarre i dati con ulteriori formati per la visualizzazione di ambienti virtuali 3D come il formato X3D

Bibliografia

[1] L. Bianchini, “Progettazione di un tool open source per l’importazione, l’esportazione e la manipolazione di dati in formato standard citygml su database postgis,” Master’s thesis, Università degli studi di Padova, Ottobre 2009.

[2] T. H. Kolbe, G. König, C. Nagel, and A. Stadler, *3DCityDB-Documentation*. Institute for Geodesy and Geoinformation Science, Technische Universität Berlin, 2009. v2.0.1.

[3] A. Stadler, C. Nagel, G. König, and T. H. Kolbe, “Making interoperability persistent: A 3d geo database based on citygml,” *Proceedings of the 3rd International Workshop on 3D Geo-Information*, p. 175, 2009.

[4] L. Calderoni, “Integrazioni di standard emergenti per la rappresentazione gis 3d,” Master’s thesis, Università di Ferrara, 2007.

[5] S. Z. Chen Tet Khuan, Alias Abdul-Rahman, “3d solids and their management in dbms,”

[6] G. Gröger, T. H. Kolbe, A. Czerwinski, and C. Nagel, *OpenGIS City Geography Markup Language (CityGML) Encoding Standard*. OGC, 2008-08-20. Version: 1.0.

[7] E. Ort and B. Mehta, “Java architecture for xml binding (jaxb).” http://www.oracle.com/technetwork/articles/javase/index-140168.html, March 2003.

[8] T. PostgreSQL, “Documentazione sulla tecnologia jdbc di postgresql.” http://jdbc.postgresql.org/documentation/84/index.html.

[9] A. Stadler and T. H. Kolbe, “Spatio-semantic coherence in the integration of 3d city models,” in *5th Internationasl Symposium on Spatial Data Quality - Modelling qualities in space and time*, 13-15 June 2007.

[10] T. Wilson, *OGC® KML*. OGC, 2008-04-14. Version: 2.2.0.

[11] “Wiki citygml4j.” http://opportunity.bv.tu-berlin.de/software/projects/citygml4j/wiki.

[12] “. geotools documentation.” http://docs.geotools.org/.

[13] PostGIS, *PostGIS 1.5.2 Manual*.

[14] “Java api per la libreria postgis.” http://www.postgis.org/documentation/javadoc/.