**Západočeská univerzita v Plzni**

**Fakulta aplikovaných věd**

**Katedra matematiky**

## Diplomová práce

# Modelování budov a terénu 3D modelu Prahy podle standardu CityGML

**Plzeň, 2014** **Jiří Fiedler**

# Prohlášení

Předkládám tímto k posouzení a následné obhajobě diplomovou práci zpracovanou na závěr magisterského studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím zdrojů uvedených v seznamu, který je její součástí.

V Plzni dne 28. 5. 2014 …….

Jiří Fiedler

# Poděkování

Na tomto místě bych rád poděkoval vedoucímu diplomové práce, Ing. Karlu Janečkovi, Ph.D. za veškeré rady a připomínky. Díky si zaslouží také má přítelkyně Radka za podporu a v neposlední řadě i přispěvovatelé webu stackoverflow.com, kteří mi ušetřili ohromné množství času nejen při psaní diplomové práce.

# Abstrakt

Práce se zabývá tvorbou 3D modelů měst z existujících dat podle standardu CityGML. Zahrnuje výběr CityGML tříd vhodných pro reprezentaci dostupných GIS dat, návrh mapování tříd těchto dat na vybrané CityGML třídy a návrh a realizaci převodu těchto dat z GIS do CityGML. V rámci práce je také popsán program FME sloužící pro konverze prostorových dat. Kromě toho byl vytvořen a pro převod použit nástroj pro identifikaci shluků polygonů.



# Klíčová slova

3D model; CityGML; identifikace objektů; FME



# Abstract

This thesis is concerned with 3D building modelling in accordance with the CityGML standard. It comprises the selection of CityGML classes suitable for available GIS data representation, design of this data classes to selected CityGML classes mapping and design and realisation of GIS to CityGML conversion of this data. Spatial data conversion software FME is also described within the scope of this thesis. Moreover, the polygons clusters identification tool has been created and used for the conversion.



# Key words

3D model; CityGML; objects identification; FME

# Obsah

# 1 Úvod

Od realizace původního zadání diplomové práce, generalizace 3D budov v souladu se standardem CityGML, bylo po studiu v seznamu odborné literatury uvedených článků upuštěno. Ukázalo se totiž, že by bylo obtížné rozšířit v článcích publikované výsledky o vlastní přínos. Pokus o kontaktování autorů nejslibněji vypadajících článků pak nebyl úspěšný.

Z těchto důvodů bylo směřování práce po dohodě s vedoucím změněno na tvorbu 3D modelů v souladu se standardem CityGML, k čemuž byla poskytnuta 3D data ve formátu shapefile (Digitální model zástavby a zeleně hlavního města Prahy).

Převod dat z ploché tabulkové struktury GIS do objektového modelu CityGML neznamená pouze konverzi z jednoho formátu souboru do jiného. Oba přístupy se liší natolik zásadně, že převod mezi těmito formáty zahrnuje mnoho úkonů s vlastním převodem často zdánlivě nesouvisejících. Zatímco v GIS je význam jednotlivých prvků dán atributy, v CityGML již objektem samotným. Atributy je možné významovou informaci obohatit, ale základní údaj, jaký prvek skutečného světa objekt popisuje, udává třída, jejíž je instancí. Geometrická a významová informace je tak mnohem těsněji svázána.

Data pro GIS sestávají z geometrických prvků (body, linie, polygony), ke kterým jsou přiřazené atributy. CityGML data se skládají z objektů odpovídajícím prvkům skutečného světa (budovy, mosty, silnice,…), ke kterým je přiřazena geometrie a případně atributy upřesňující jejich vlastnosti. Ještě výrazněji objektový přístup vyniká možností popsat hierarchické vztahy mezi objekty (okno je součást zdi, zeď je součást budovy) bez pomoci geometrie.

Cílem této práce je prozkoumat datový model CityGML, vybrat z něj třídy vhodné pro reprezentaci Digitálního modelu zástavby a zeleně hlavního města Prahy a navrhnout a realizovat postup, jak tuto ve formátu shapefile uloženou datovou sadu převést do CityGML.

Práce je členěna do několika kapitol. V té následující je popsána pro Digitální model zástavby a zeleně hlavního města Prahy relevantní podmnožina datového modelu CityGML. Kapitola 3 popisuje použitá data a věnuje se výběru CityGML tříd pro jejich reprezentaci. Kapitola 4 pak pojednává o identifikaci objektů ve zdrojových datech, nutném předpokladu pro převod do CityGML, kterému je věnována kapitola poslední.

# 2 CityGML

CityGML je otevřený vektorový formát pro reprezentaci virtuálních 3D modelů měst založený na XML. Je to aplikační schéma pro GML3 (Geography Markup Language, verze 3), výměnný XML formát pro prostorová data. Oba formáty jsou definovány mezinárodním konsorciem pro geodata, OGC (Open Geospatial Consortium). Od jiných souborových formátů pro ukládání 3D dat se odlišuje tím, že kromě geometrie zahrnuje i sémantiku, atributy a topologické vazby, což rozšiřuje možnosti jeho využití od prosté vizualizace k prostorovým analýzám. V této kapitole jsou popsány aspekty CityGML podstatné pro tuto práci. Veškeré informace jsou čerpány z [1], odkud pochází i všechny obrázky kromě obr. 2.8.

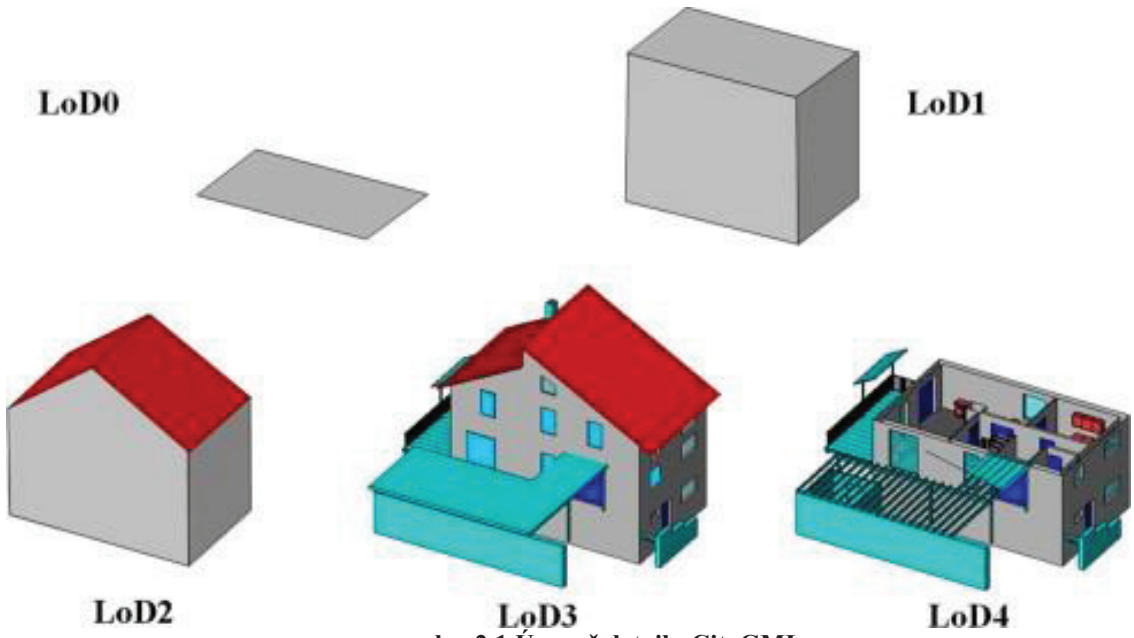
## 2.1 Víceúrovňový model

CityGML umožňuje uložení jednoho objektu v různých úrovních detailu současně. Díky tomu například může vizualizační aplikace zobrazovat objekt různě v závislosti například na vzdálenosti objektu od kamery.

Úrovně detailu nejsou určeny zcela striktně, v zásadě jsou charakterizovány minimálními rozměry objektů, které se v konkrétní úrovni mohou objevit, ale také dalšími vlastnostmi. Definováno je 5 úrovní detailu (LOD – Level Of Detail):

- LOD0 coby dvou a půl dimenzionální model, v případě budov půdorys nebo obrys střechy
- LOD1 obsahující objekty od velikosti půdorysu 6 × 6 m v podobě hranolů
- LOD2 s objekty již od velikosti půdorysu 4 × 4 m, v případě budov včetně střech i s přesahy
- LOD3 obsahující všechny vnější detaily včetně oken a dveří, objekty již od velikosti půdorysu 2 × 2 m
- LOD4 obsahující navíc i prvky interiéru

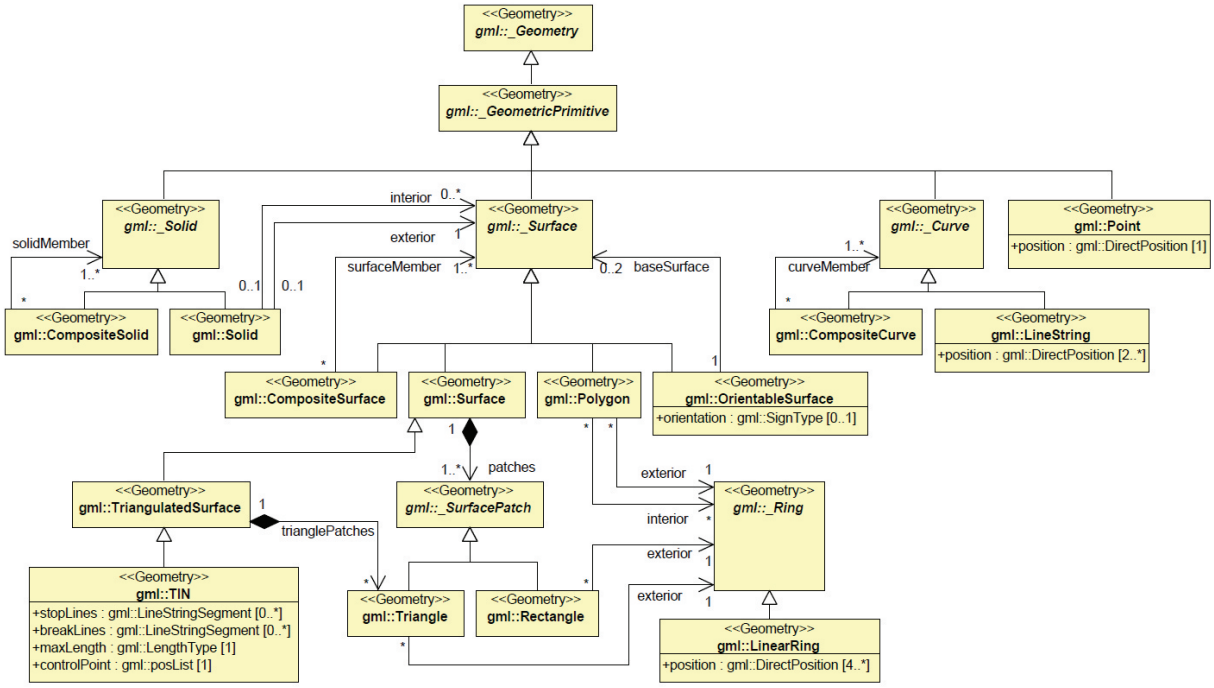
Úrovně detailu ilustruje obr. 2.1. Na objekty všech úrovní je možné přidat vzhled, například textury.




**obr. 2.1 Úrovně detailu CityGML**

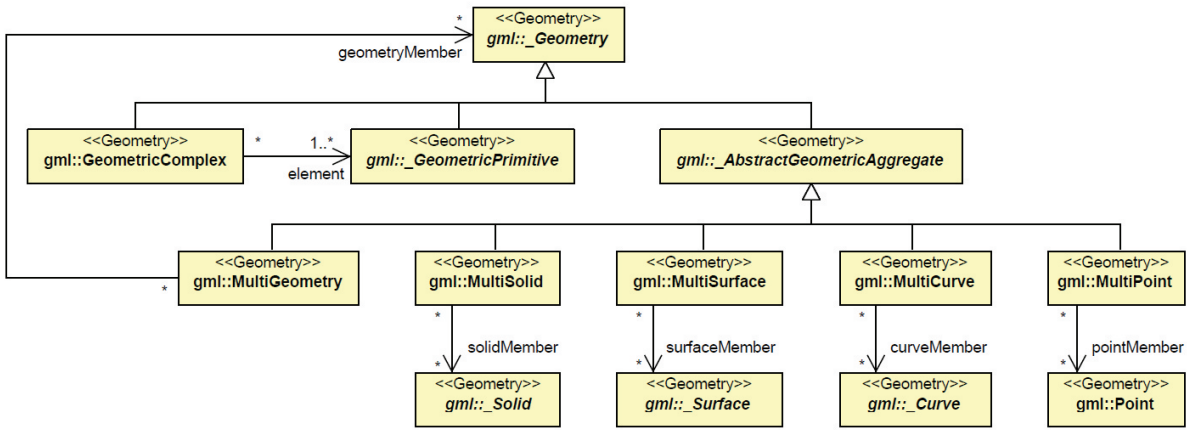
## 2.2 Prostorový model

CityGML využívá geometrický model GML3, který geometrii ukládá podle hraniční reprezentace, kdy je těleso tvořeno hraničními plochami a plocha hraničními křivkami. Využita je pouze podmnožina geometrického modelu GML3, kterou ilustruje obr. 2.2 a obr. 2.3. Nejsou podporovány křivky, takže všechny objekty *gml::_Curve* musejí být typu `gml::LineString`. Obdobně pro plochy jsou přípustné pouze polygony.




**obr. 2.2 Geometrický model CityGML – primitiva a složené prvky**

V CityGML existuje více způsobů, jak popsat prostorový objekt složený z více částí. Jednou z možností je složený prvek (`CompositeCurve`, `CompositeSurface` a `CompositeSolid` na obr. 2.2). Ten může obsahovat pouze prvky stejné dimenze, které musí být disjunktní a topologicky spojené. Volnější způsob je komplex, jehož prvky musí být disjunktní a mohou, ale nemusejí, se dotýkat. Komplex může obsahovat prvky různé dimenze. Zcela volné prostorové uspořádání platí pro agregace (`MultiPoint`, `MultiCurve`, `MultiSurface` a `MultiSolid` na obr. 2.3), které slouží v podstatě jako množina primitiv.




**obr. 2.3 Geometrický model CityGML – komplexy a agregace**

Sdílené části geometrie mohou být uloženy pouze jednou. Z prvků, které tuto část obsahují, je pak na ni odkazováno pomocí *XLinks*. Může se jednat o sdílení stejné geometrie dvěma různými prvky (např. různé produktovody se stejnou trasou), sdílení geometrie mezi prvkem a geometrií jiného prvku (zeď je zároveň samostatný prvek a zároveň součást budovy) a případ, kdy jedna geometrie je hraniční součástí dvou jiných (např. zeď společná dvěma budovám).

Kromě sdílení částí geometrie podporuje CityGML ještě tzv. implicitní geometrii. Když se nějaký objekt v oblasti pokryté datovou sadou opakuje (například semafor, lavička, přístřešek na zastávce veřejné dopravy,…), je možné uložit jeho geometrii pouze jednou, v místním souřadnicovém systému. Na všech ostatních místech je pak pouze odkaz na tuto geometrii spolu se souřadnicemi bodu, na kterém se má objekt zobrazit, a transformační maticí sloužící pro případnou rotaci, změnu měřítka a místní posun objektu. Geometrii objektu lze uložit přímo v datové sadě CityGML prostřednictvím geometrického objektu GML3, nebo v externím souboru v proprietárním formátu (VRML, DXF,…).

## 2.3 Tematický model

Tematický model je definován množinou XML schémat, která popisují jednotlivé moduly (budovy, mosty, reliéf,…). Každý modul tak má vlastní jmenný prostor a v datových souborech je jasně patrné, do kterého modulu jaký objekt patří. V CityGML 2 je definováno 14 modulů:

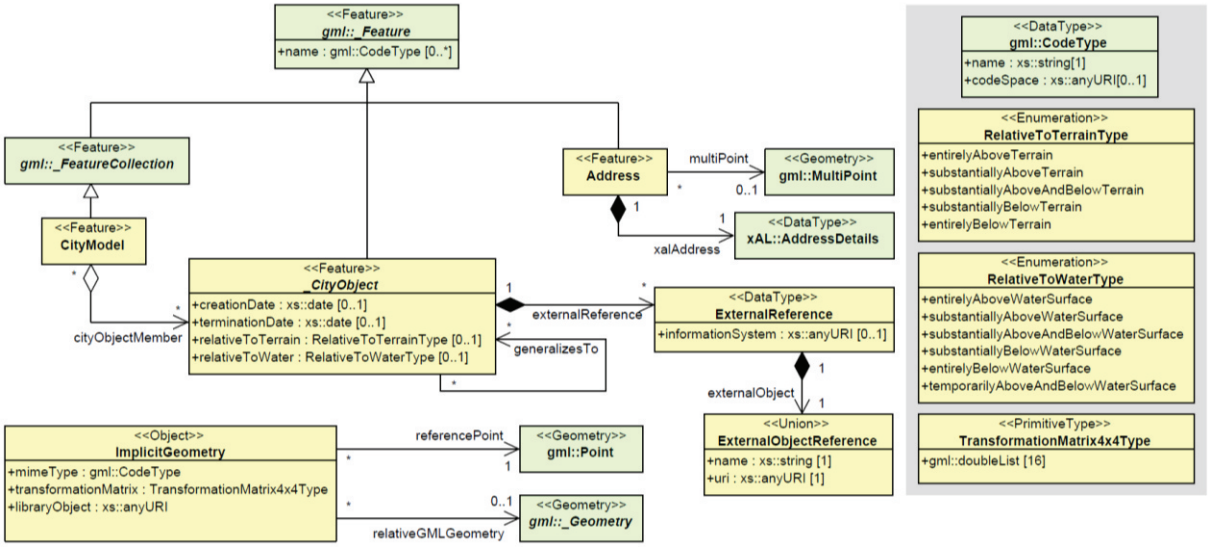
- CityGML Core
  - definuje základní komponenty datového modelu – základní abstraktní třídy a třídy společné pro více modulů.
- Vzhled
  - poskytuje prostředky pro modelování nejen vzhledu prvků, ale i jiných vlastností charakteristických pro povrch objektů
  - může sloužit například pro hlukové mapy, znázornění úniků tepla, apod.
  - využitelný nejen při vizualizaci, ale i pro analýzy nad modelem
  - v modelu je možné uložit libovolné množství témat, a to v různých úrovních detailu
- Mosty
- Budovy
- Městský mobiliář
  - lampy, lavičky, zastávky,…
- Seskupení objektů
  - mohou být tvořena libovolnými objekty, včetně dalších seskupení
  - mohou mít vlastní atributy
  - každý prvek skupiny může mít roli
- Generics
  - obecná rozšíření datového modelu
  - pro prvky, pro které není jiný vhodný modul
- Využití půdy
- Reliéf
  - rastr, TIN, zlomové linie nebo mračna bodů
- Doprava
  - silnice, železnice, náměstí,…
  - liniová síť nebo geometrický popis povrchu

- Tunely
- Vegetace
  - oblasti i např. jednotlivé stromy
- Vodní plochy
- Struktura povrchu (*deprecated* – v dalších verzích bude tento modul odebrán, místo něj by se měl používat modul pro vzhled)

Aplikace mohou podporovat libovolnou množinu rozšiřujících modulů, její součástí však vždy musí být hlavní modul (CityGML Core). Zvolená kombinace modulů se nazývá profil. Profil může být definován buďto přímo v CityGML dokumentu množinou použitých schémat modulů, anebo vlastním souborem schématu, kde jsou schémata použitých modulů importována. Tato práce se zabývá modelováním budov a terénu, v následujících podkapitolách jsou tedy popsány moduly CityGML Core, Reliéf a Budovy.

### 2.3.1 CityGML Core

Modul CityGML Core je nedílnou součástí každého profilu. Jsou v něm definovány základní abstraktní třídy, ze kterých jsou odvozené tematické třídy rozšiřujících modulů, a základní datové typy a tematické třídy společné pro více modulů. UML diagram modulu je na obr. 2.4. Třídy patřící do CityGML Core jsou podbarveny žlutě. Zelené podbarvení náleží třídám pocházejícím z GML3. Jména abstraktních tříd jsou psána kurzívou a začínají podtržítkem.



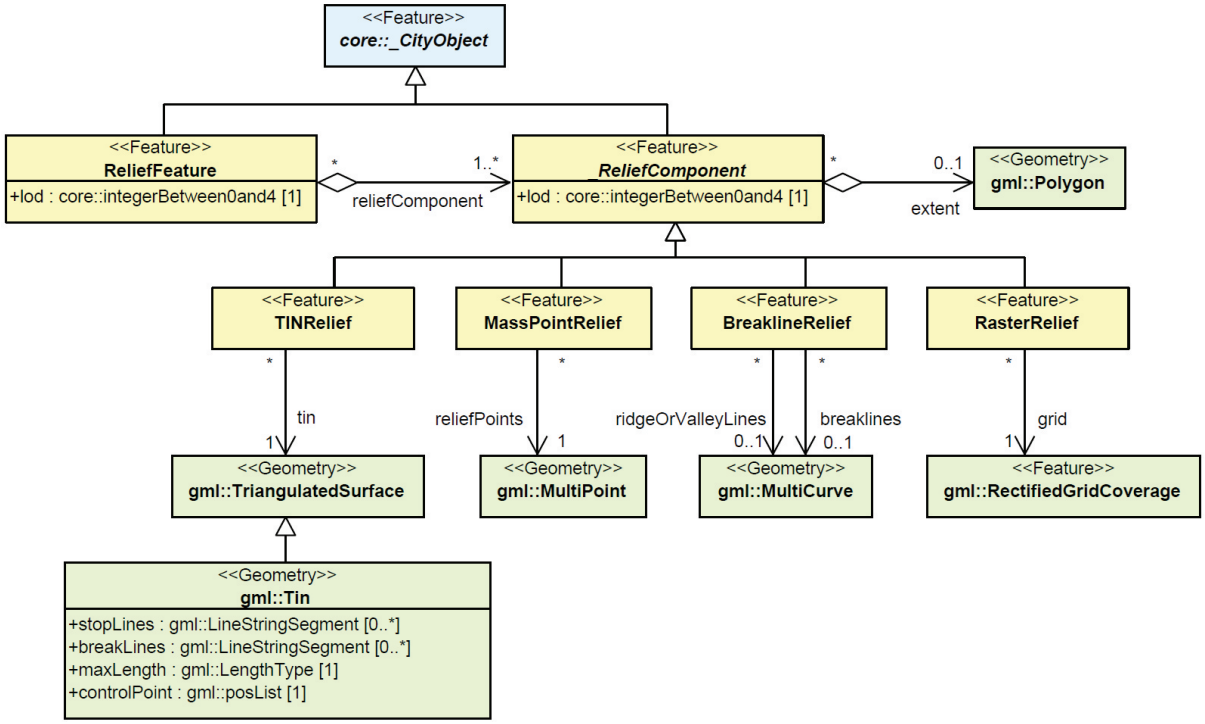

**obr. 2.4 UML diagram modulu CityGML Core**

Všechny tematické třídy v CityGML jsou odvozeny z abstraktní třídy `_CityObject`. Obsahuje tedy atributy společné pro jednotlivé tematické třídy. Atributy `relativeToTerrain`, respektive `relativeToWater`, popisují pozici objektu vzhledem k terénu, respektive hladině vody, i bez přítomnosti digitálního modelu terénu, respektive vodních ploch, v datové sadě. Vazba `externalReference` slouží k případné identifikaci objektu v jiné datové sadě. Vazba `generalizesTo` umožňuje uložení objektu v různých úrovních detailu. Směřuje od vyšší úrovně k nižší. Více různých objektů vyšší úrovně detailu může mít společného reprezentanta v nižší úrovni detailu. Třída `CityModel` slouží ke sdružování objektů. Použití třídy `ImplicitGeometry` je pospáno v kapitole 2.2.

### 2.3.2 Reliéf

Digitální model reliéfu může být v CityGML uložen jako rastr, TIN, mračno bodů, či prostřednictvím zlomových linií. TIN může být reprezentována třídou `TrianguledSurface` nebo `Tin`. V případě `TrianguledSurface` jsou uloženy přímo trojúhelníky sítě, při použití třídy `Tin` pak pouze body spolu se zlomovými liniemi, vlastní TIN je pak třeba vygenerovat např. Delaunayovou triangulací.

V jedné datové sadě je možné různé reprezentace reliéfu kombinovat. Přípustná je jak různá reprezentace jednotlivých úrovní detailu stejného území, tak použití odlišné reprezentace pro různé části území. Rozsah platnosti každé části reprezentace reliéfu je pak dán polygonem. Je také možné popsat reliéf dané oblasti a úrovně detailu více způsoby současně. Příkladem může být rastr doplněný zlomovými liniemi. UML diagram modulu je na obr. 2.5. Třídy modulu Reliéf jsou podbarveny žlutě, zelené podbarvení opět náleží třídám pocházejícím z GML3.

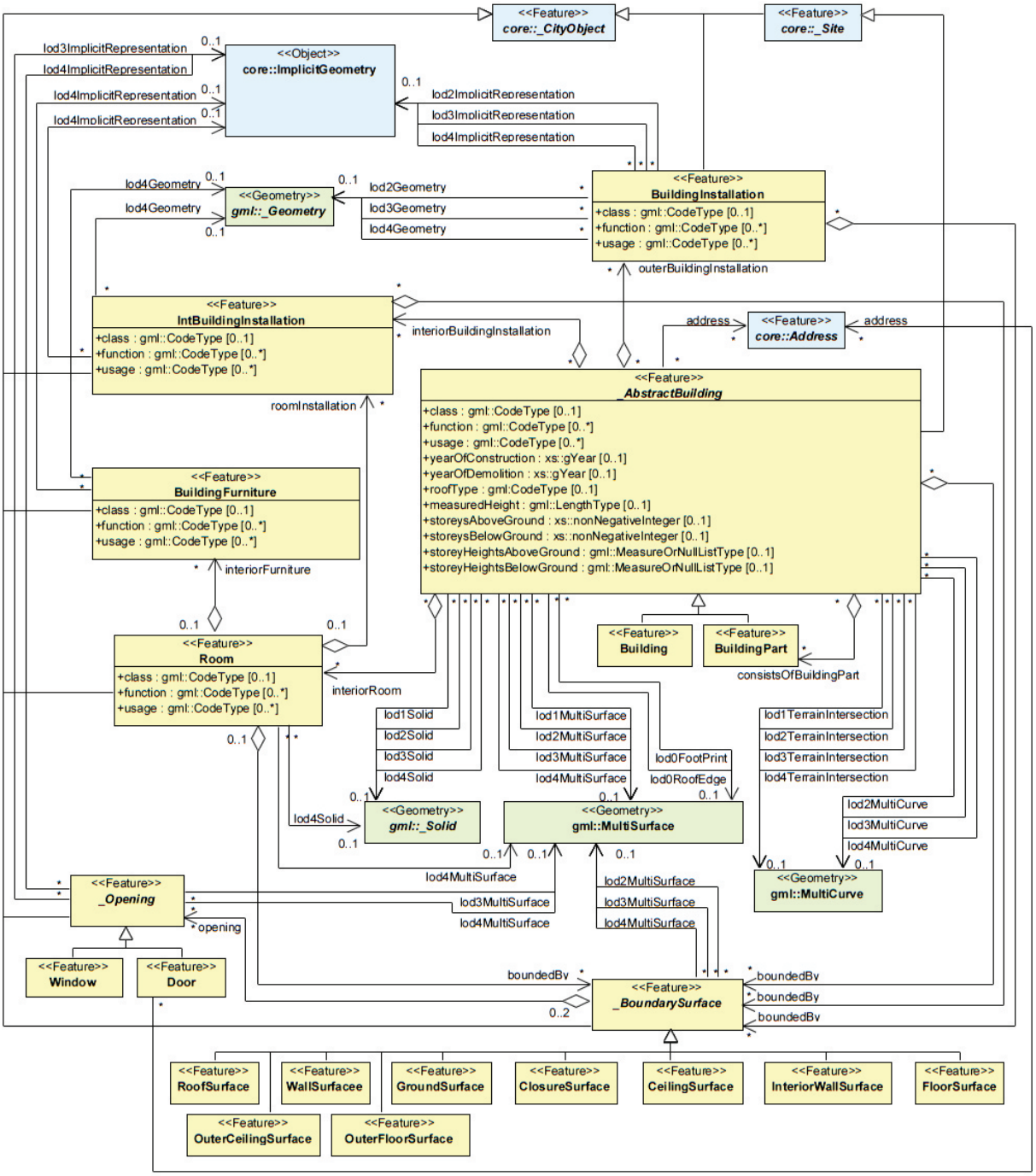



**obr. 2.5 UML diagram digitálního modelu reliéfu v CityGML**

Úroveň detailu je přítomna jak u třídy `ReliefFeature`, tak u třídy `_ReliefComponent`. Mohou totiž nastat situace, kdy je vhodné popsat úroveň detailu celého reliéfu rozdílně od jeho některých částí. Typicky k tomu dojde tehdy, když máme k dispozici reliéf v nižší úrovni detailu (např. rastr v LOD2) pro celé území pokryté datovou sadou, a k tomu pro některé oblasti reliéf podrobnější (např. TIN v LOD3). Pro zobrazení v nižší úrovni detailu bude sloužit rastr v LOD2 (označení jak celku, tak komponent; v tomto případě bude pravděpodobně komponenta jediná). Pro detailnější zobrazení pak poslouží ten samý rastr, který však bude v oblastech, kde je k dispozici i podrobnější TIN, tímto nahrazen. Jednotlivé komponenty budou označeny příslušným číslem LOD, celek ponese označení LOD3. Ohraničení jednotlivých částí definují polygony rozsahu platnosti. Při vizualizaci se tak dá snadno zabránit překrývání jednotlivých reprezentací, stačí nezobrazovat část DMR, která leží mimo polygon.

### 2.3.3 Budovy

Budovy je možné v CityGML uložit v pěti úrovních detailu. Je podporováno členění budov na části. To se hodí v případě, kdy je budova složena z více dílů lišících se v parametrech, které by byly standardně vlastní budově (například počet pater nebo typ střechy). Části budov pak mohou být členěny dále. UML diagram modulu je na obr. 2.6. Třídy modulu Budovy jsou podbarveny žlutě, třídy z CityGML Core modře, zelené podbarvení opět náleží třídám pocházejícím z GML3.




**obr. 2.6 UML diagram pro modelování budov v CityGML**

Základem je třída `_AbstractBuilding`, která obsahuje geometrii i vlastnosti budov a jejich částí. Atributy `class`, `function` a `usage` nabývají číselných hodnot, jejichž význam definují číselníky. CityGML obsahuje návrhy vhodných číselníků, ale je možné použít vlastní. Atribut `class` slouží k základnímu roztřídění budov, tedy zda se jedná o budovu obytnou,

rekreační, veřejnou, atd. Atribut `function` popisuje zamýšlený účel budovy (kostel, obchod, hotel,…), atribut `usage` pak její skutečné využití (může nabývat stejných hodnot jako `function`). Atributy popisující střechu budovy, její výšku a patra mohou sloužit pro vykreslení budovy, ke které není k dispozici 3D geometrie.

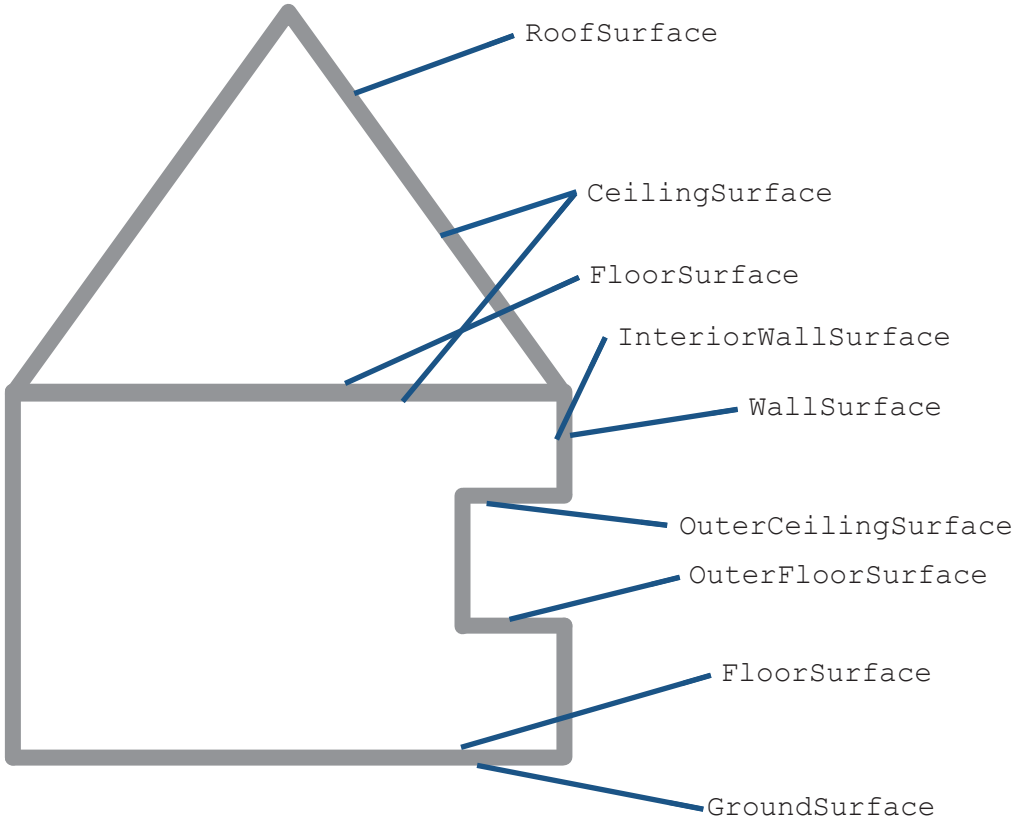
Vazby `lodXTerrainIntersection` slouží k popisu průsečnice povrchu budovy s terénem. Na obr. 2.7 je vyznačena černě. Je-li tato křivka v datech přítomna, může být digitální model terénu při vizualizaci upraven tak, aby protínal model budovy ve správném místě. Kromě toho slouží i ke správnému umístění textur na model budovy. Protože geometrie budovy může být pro různé úrovně detailu různá, je i tato průsečnice definována zvlášť pro každou úroveň vyšší než 0.

**obr. 2.7 Průsečnice budovy s terénem**

Třída `BuildingInstallation` slouží k modelování takových prvků budovy, které není vhodné popisovat jako část budovy, příkladem jsou například komíny, schodiště, balkony apod. Její atributy `class`, `function` a `usage` mají obdobný význam jako tytéž atributy u budov.

Budova (nebo její část) může obsahovat místnosti reprezentované třídou `Room`. Pohyblivé věci v místnostech (nábytek) popisuje třída `BuildingFurniture`, ty nepohyblivé pak třída `IntBuildigInstallation`. Protože mnohé nepohyblivé prvky budov nejsou spojeny s jedinou místností, může být třída `IntBuildingInstallation` asociována přímo s budovou. Příkladem může být schodiště či zábradlí.

Třída *`_BoundarySurface`* slouží, prostřednictvím podtříd pro jednotlivé typy povrchu, k popisu významu jednotlivých povrchů budov a jejich částí, místností a vnějších i vnitřních prvků budov. Použití jednotlivých podtříd ilustruje obr. 2.8. Bližší vysvětlení si zaslouží třída `ClosureSurface`, která slouží jako nepravý povrch. Je možné ji použít například pro oddělení 2 místností, mezi kterými není zeď (např. kuchyně spojená s obývacím pokojem), či

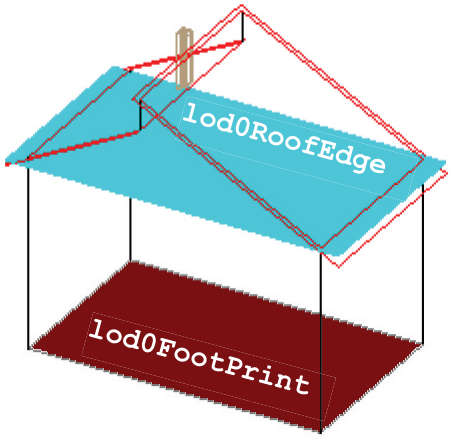



**obr. 2.8 Použití různých tříd reprezentujících typy povrchů**

uzavřít jinak částečně otevřené budovy. Tento pomocný povrch se nezobrazí, umožní však výpočty objemů místností či budov, které by jinak nebyly celé ohraničené.

Povrchy mohou obsahovat dveře a okna geometricky reprezentované dírou v polygonu povrchu (děravý polygon má vnější a vnitřní obvod, viz vazba `interior` mezi `gml::Polygon` a *`gml::_Ring`* na obr. 2.2). Body vnitřního obvodu polygonu povrchu pak mohou být tytéž jako body vnějšího obvodu dveří, respektive okna, které společně reprezentuje třída *`_Opening`*.

Jak je vidět z diagramu na obr. 2.6, jsou mnohé části modelu budovy přítomny až od určité úrovně detailu. V LOD0 je budova reprezentována pouze vodorovným polygonem. Dvě vazby jsou v diagramu proto, že je možné uložit zvlášť půdorys budovy bez střechy (`lod0FootPrint`) a zvlášť půdorys střechy (`lod0RoofEdge`), který bývá díky přesahům větší, což ilustruje obr. 2.9. V obou případech jsou pro popis geometrie použity 3D souřadnice, výška všech vrcholů každého z polygonů však musí být stejná. V LOD1 je budova reprezentována jednoduchými bloky. Ty je možné popsat buďto objemovým modelem (`lod1Solid`), nebo jako množinu povrchů (`lod1MultiSurface`).




**obr. 2.9 Reprezentace budovy v LOD0**

Od LOD2 výše je možné oba předchozí přístupy modelování geometrie kombinovat. Navíc se využívá třída `BuildingInstallation` sloužící k popisu menších částí budovy. Samotná budova pak může být obohacena o liniové prvky třídou `gml::MultiCurve`.

Jednotlivým částem povrchu budovy může být přiřazen význam, a to prostřednictvím podtříd třídy *_BoundarySurface*. Je-li tomu tak, geometrická reprezentace budovy musí namísto explicitní definice geometrie ukazovat pomocí `XLink` na geometrie jednotlivých prvků povrchu, aby nedocházelo ke zbytečnému duplikování uložení geometrie. Geometrie povrchů nesmí v LOD2 obsahovat otvory. V LOD3 se přidává třída *_Opening* pro popis oken a dveří, v LOD4 pak třídy sloužící k modelování interiérů (`Room`, `BuildingFurniture` a `IntBuildingInstallation`).

## 2.4 XML reprezentace

Datový model CityGML je v XML realizován tak, že každé třídě a každé agregační vazbě odpovídá element XML dokumentu s příslušným názvem včetně velikosti písmen. Jména elementů odpovídajících třídám tak začínají velkým písmenem, zatímco jména elementů odpovídajících vazbám písmenem malým. Následuje ukázka z GML souboru s terénem, který se coby jeden z výstupů této práce nachází celý na přiloženém CD.

```
<cityObjectMember>
   <dem:TINRelief gml:id="TINRelief1">
      <dem:tin>
         <gml:TriangulatedSurface srsDimension="3">
            <gml:trianglePatches>
               <gml:Triangle>
                  <gml:exterior>
                     <gml:LinearRing>
                        <gml:posList>
                           -744999.234  -1042004.812   270.819
                           -745000      -1042000       270.625
                           -745000      -1042004.045   270.812
                           -744999.234  -1042004.812   270.819
                        </gml:posList>
                     </gml:LinearRing>
                  </gml:exterior>
               </gml:Triangle>
               <gml:Triangle>
                  <gml:exterior>
                     <gml:LinearRing>
```

Dále v souboru následují další trojúhelníky tvořící TIN.

Element `cityObjectMember` odpovídá vazbě mezi třídami `CityModel` a *_CityObject* na obr. 2.4. Konkrétní potomek této abstraktní třídy je dán elementem `dem:TINRelief`, který je spolu s vazbou `dem:tin` a třídou `gml:TriangulatedSurface` na obr. 2.5. Protože jsou v souboru uloženy přímo trojúhelníky sítě, není použita třída `gml:Tin` a pro nalezení tříd a vazeb odpovídajících dalším vnořeným elementům na ukázce je třeba se podívat na obr. 2.2. Na něm je vazba `gml:trianglePatches` spojující `gml:TriangulatedSurface` se třídou `gml:Triangle`, ze které pak vede vazba `gml:exterior`. Třída *gml:_Ring* je abstraktní, použita je tedy třída `gml:LinearRing`, která už obsahuje přímo souřadnice lomových bodů jednotlivých trojúhelníků v elementu `gml:posList`.

Elementy odpovídající třídám a vazbám CityGML lze sledovat i v dokumentu budov. Následuje jeho krátká ukázka obsahující jeden polygon z vikýře. Celý GML soubor se opět nachází na přiloženém CD.

```
<cityObjectMember>
   <bldg:Building gml:id="budova901001780">
      <bldg:outerBuildingInstallation>
         <bldg:BuildingInstallation gml:id="shluk19">
            <bldg:boundedBy>
               <bldg:WallSurface gml:id="fme-gen-h3155...c33">
                  <bldg:lod2MultiSurface>
                     <gml:MultiSurface srsName="5514" srsDimension="3">
                        <gml:surfaceMember>
                           <gml:Polygon>
                              <gml:exterior>
                                 <gml:LinearRing>
                                    <gml:posList>
                                       -743353.877  -1043725.74  212.266
                                       -743353.877  -1043725.74  214.606
                                       -743333.755  -1043725.197 214.606
                                       -743333.755  -1043725.197 212.266
                                       -743353.877  -1043725.74  212.266
                                    </gml:posList>
                                 </gml:LinearRing>
                              </gml:exterior>
                           </gml:Polygon>
                        </gml:surfaceMember>
                        <gml:surfaceMember>
                           <gml:Polygon>
                              <gml:exterior>
```

Stejně jako v případě DMR začíná ukázka elementem `cityObjectMember`, jehož jediným předkem v XML dokumentu je kořenový element `CityModel`. Třída `bldg:Building`, která je potomkem *`core:_CityObject`*, se nachází na obr. 2.6. Dále se nacházející element `bldg:outerBuildingInstallation` odpovídá vazbě mezi třídami *`bldg:_AbstractBuilding`* a `bldg:BuildingInstallation` a na obr. 2.6 lze vypozorovat i vazbu `bldg:boundedBy` na třídu *`bldg:_BoundarySurface`*, která je abstraktním rodičem třídy `bldg:WallSurface`, jakož i vazbu `bldg:lod2MultiSurface` směřující ke třídě `gml:MultiSurface`. Odsud vedoucí vazbu odpovídající elementu `gml:surfaceMember` je již nutné hledat na obr. 2.3. Vede k abstraktní třídě *`gml:_Surface`*, jejíchž potomků je použita třída `gml:Polygon`. Ta už je, stejně jako vazba `gml:exterior` na třídu `gml:LinearRing`, vidět na obr. 2.2.

V GML souboru pak následují další elementy `gml:surfaceMember` tvořící dohromady agregovanou geometrii zdí vikýře. Dále pak tento vikýř (tedy objekt třídy `bldg:BuildingInstallation`) obsahuje obdobným způsobem strukturovaný objekt třídy `bldg:RoofSurface`, který je s ním svázán vazbou `bldg:boundedBy`.

# 3 Použitá data

Podkladem pro tuto práci je výřez z Digitálního modelu zástavby a zeleně hlavního města Prahy ve formátu SHP. Výřez pokrývá oblast Starého Města, Hradčan a Malé Strany a zasahuje až ke strahovským kolejím. Obsahuje tak rozličné typy budov a zajímavý terén, data jsou tedy vhodná pro testování 3D modelování. Ukázka je na obr. 3.1. Souřadnicový systém je ESRI verze S-JTSK (osa X orientovaná na sever, osa Y na západ; souřadnice pro území Česka jsou tedy záporné a oproti klasickému S-JTSK prohozené).

**obr. 3.1 Digitální model zástavby a zeleně hlavního města Prahy**

## 3.1 Datový model

Datová sada obsahuje vrstvy pro budovy, terén a vegetaci; pro převod do CityGML byly vybrány jen budovy a terén. Terén je reprezentován trojúhelníkovou sítí, která je uložena ve formě jednotlivých trojúhelníků coby polygonů v 3D SHP, nikoliv jako TIN. Budovy jsou reprezentovány svým povrchem, je tedy použita hraniční reprezentace. Každá budova se skládá z mnoha polygonů reprezentujících její stěny, střechu a další prvky. Tyto polygony jsou rozděleny do celkem osmi tříd:

- svislá obvodová stěna
- šikmá střešní plocha
- vodorovná střešní plocha
- dílčí plocha střešní kruhové plochy
- vikýř, střešní nadstavba
- význačná věž na střeše
- komín
- výtah, větrání, klimatizace

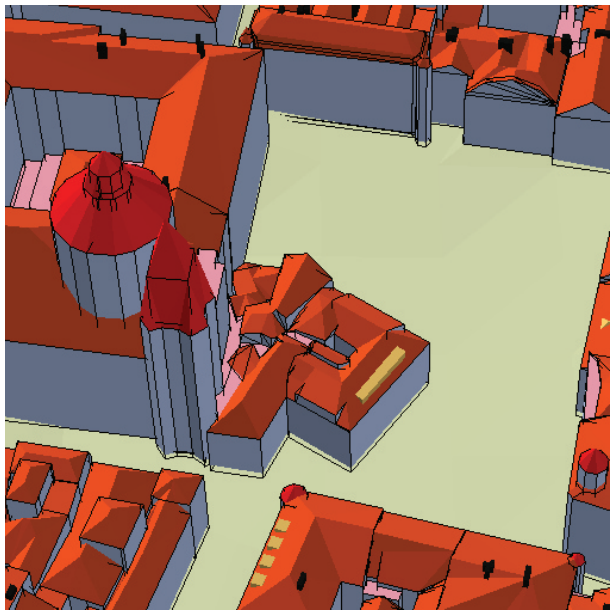
Stěny společné dvěma sousedním budovám jsou uloženy duplicitně, každá budova je jako celek obklopena plochami zdí a střech. Odspodu budovy uzavřeny nejsou. Datová sada budov obsahuje i čísla budov. Lze tak snadno určit, ke které budově patří který polygon. [2]

## 3.2 Podrobnost dat

Zdrojem dat digitálního modelu zástavby a zeleně je letecké snímkování. Tím je dán jeho obsah a také podrobnost. Fotogrammetrickým vyhodnocováním jsou určeny výšky, obvody a tvar střešního pláště budov. Skutečné tvary stěn a prvky fasád nejsou modelovány, půdorysy budov odpovídají obvodu střešního pláště. [3]

Ze způsobu pořízení dat vyplývá, jaké prvky budov jsou jejich součástí. Neobsahují například podloubí či balkony, naopak komíny, vikýře a různé věžičky na střechách modelovány jsou. Na následujících obrázcích je pro porovnání ukázáno Malostranské náměstí na leteckém snímku z webu Mapy.cz (vlevo, obr. 3.2) a v modelu zástavby a zeleně (vpravo, obr. 3.3).

**obr. 3.2 Malostranské náměstí na šikmém leteckém snímku z portálu Mapy.cz**



**obr. 3.3 Malostranské náměstí v digitálním modelu zástavby a zeleně hlavního města Prahy**

Množina prvků obsažených v digitálním modelu zástavby a zeleně odpovídá LOD2 v CityGML. Všechny prvky kromě stěn a střech totiž v CityGML spadají do třídy `BuildingInstallation`, která je v modelu budov přítomna právě již od LOD2. I význam jednotlivých polygonů může být, prostřednictvím podtříd třídy `_BoundarySurface`, modelován již v LOD2. Ze zdrojových dat je tedy možné vytvořit téměř úplnou 2. úroveň detailu podle standardu CityGML, chybět budou pouze prvky na fasádách budov.

Některé prvky digitálního modelu zástavby a zeleně (typicky komíny) však nedosahují minimální velikosti pro zanesení do LOD2 a měly by tak být generalizovány vypuštěním. Nabízejí se 3 možné přístupy:

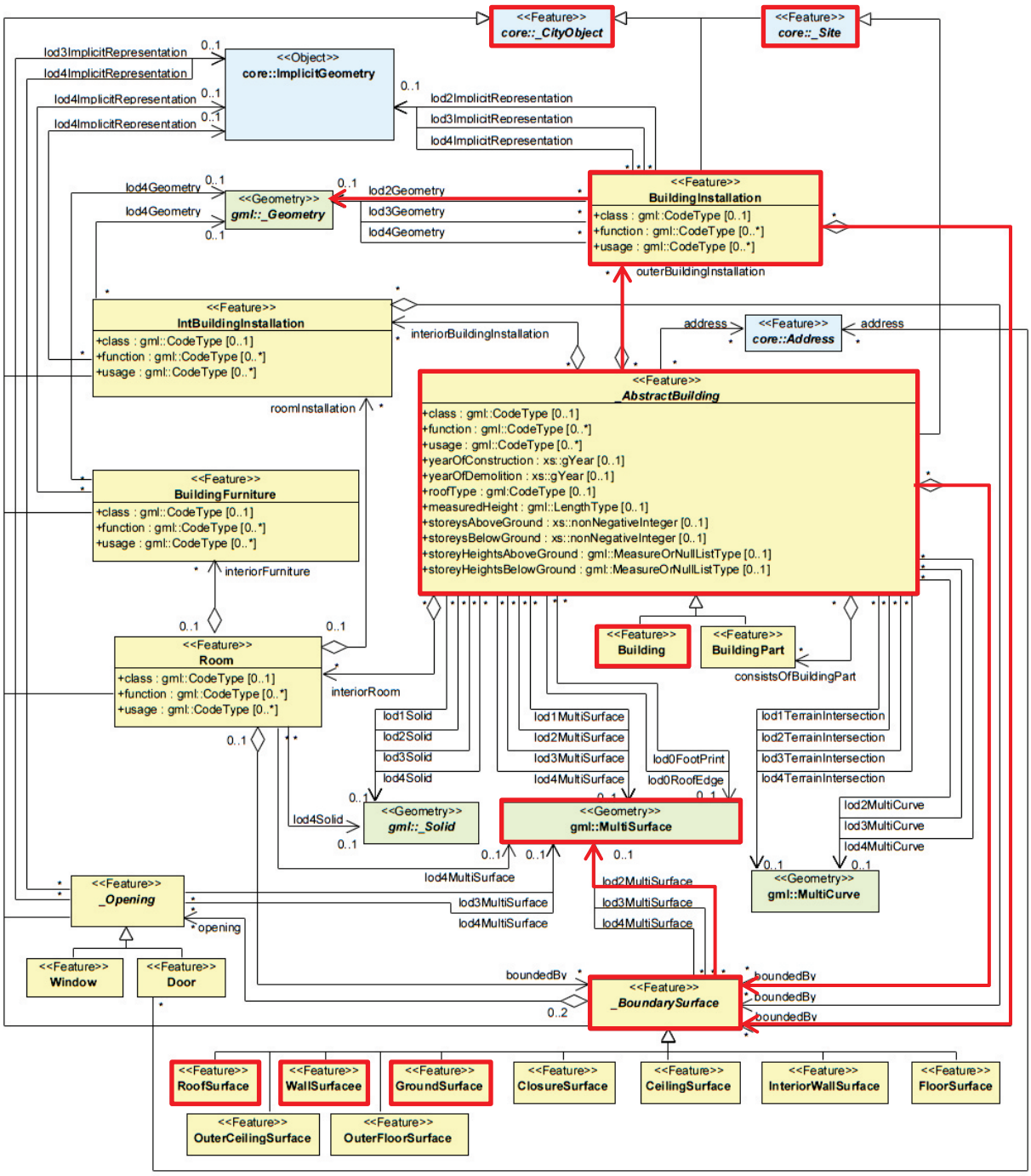
- vytvořit pouze čistou LOD2 a menší komíny do modelu v CityGML nezahrnout,
- vytvořit LOD2 s tím, že bude oproti specifikaci obsahovat i komíny podlimitní velikosti,
- kromě LOD2 vytvořit ještě LOD3, které by oproti LOD2 obsahovalo navíc pouze tyto menší komíny.

Z těchto variant byla vybrána druhá možnost, aby byla využita veškerá dostupná data a nebylo přitom nutné ukládat další úroveň detailu jen kvůli komínům.

## 3.3 Výběr CityGML tříd vhodných pro zdrojová data

Ne všechny třídy popsané v kapitole 2.3 jsou vždy využívány. Záleží na tom, jaká data modelujeme. Při převodu z jiného formátu do CityGML pak na tom, jaká data máme k dispozici.

V případě terénu máme množinu polygonů tvořící trojúhelníkovou síť pokrývající celé území. Z CityGML modulu pro reliéf tedy využijeme třídu `TinRelief` a geometrie bude reprezentována třídou `TrianguledSurface` z GML3. Pro budovy je situace složitější,




**obr. 3.4 Využité třídy CityGML modulu pro budovy**

využité třídy totiž záleží na úrovni podrobnosti dat. Na obr. 3.4 jsou proto červeně zvýrazněny třídy a agregační vazby CityGML tematického modulu pro budovy využité v této práci.

Při převodu dat do CityGML nelze vytvořit mapování vstupních tříd na výstupní 1:1. Je totiž třeba rozhodnout nejen to, jaká podtřída třídy `_BoundarySurface` bude použita pro kterou třídu vstupních dat, ale také například to, ke které třídě bude tento `_BoundarySurface` přiřazen vazbou `BoundedBy`. Schéma na obr. 3.5 nastiňuje mapování tříd vstupních dat na třídy CityGML pro budovy. V prostředním sloupci je klasifikace Digitálního modelu zástavby a zeleně, vlevo pak odpovídající třídy pro reprezentaci objektů a vpravo třídy pro reprezentaci povrchu těchto objektů.


**obr. 3.5 Schéma mapování tříd vstupních dat na třídy CityGML pro budovy**

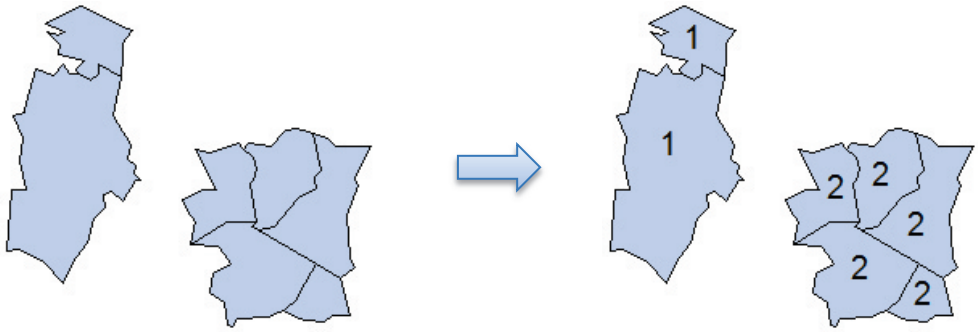
Zatímco mapování na budovy (třída `Building`), respektive prvky na nich (třída `BuildingInstallation`), je poměrně zřejmé, volba vhodných tříd pro reprezentaci povrchu je složitější. Svislé obvodové stěně bude pochopitelně přiřazena třída `WallSurface`, střechám pak třída `RoofSurface`. Náročnější situace nastává v případě vikýřů. Ty lze ve vstupních datech snadno odlišit od budov, protože jsou jim atributově přiřazeny příslušné třídy. Rozhodnutí, jaký jim přiřadit typ povrchu, však bude moci být učiněno jen na základě jejich geometrie. Svislým plochám vikýřů bude přiřazena třída `WallSurface`, ostatním `RoofSurface`.

Ke třídě `GroundSurface` nevede žádná vazba, protože budovy v digitálním modelu zástavby a zeleně nejsou zespodu uzavřeny. Protože v CityGML by objekty uzavřeny být měly, je třeba polygony zespodu budov přidat. Těmto přidaným polygonům pak bude přiřazena třída `GroundSurface`.

# 4 Identifikace objektů

Standard CityGML podporuje hierarchickou reprezentaci objektů – budovy (které navíc mohou být sdruženy do skupin) se člení na části a další objekty (komíny, vikýře,…). Každý takový objekt pak má svou geometrii. Digitální model zástavby a zeleně je složen z polygonů, jejichž význam je dán atributy. Hierarchickou strukturu však nelze přímo vyčíst. Pro každý polygon je zřejmé, zda se jedná o součást střechy, komínu, či vikýře, nelze ale z atributů rozlišit, které polygony tvoří společný objekt. Mezi atributy je k dispozici identifikátor budovy jako celku, identifikátory objektů nižších úrovní je třeba před převodem do CityGML dodat.

Jednotlivé objekty nižší úrovně (komíny, vikýře,…) lze rozlišit na základě geometrie. Analýzou sousednosti je možné zjistit, které polygony spolu tvoří shluky sousedů, a označit je společným identifikátorem. K tomuto účelu byl vytvořen Python skript, který do atributové tabulky zdrojové třídy prvků přidá sloupec s identifikátory shluků. Při následném převodu do CityGML pak tento sloupec poslouží k tvorbě složených prvků obdobně jako sloupec s identifikátorem budovy. Problém ilustruje obr. 4.1.



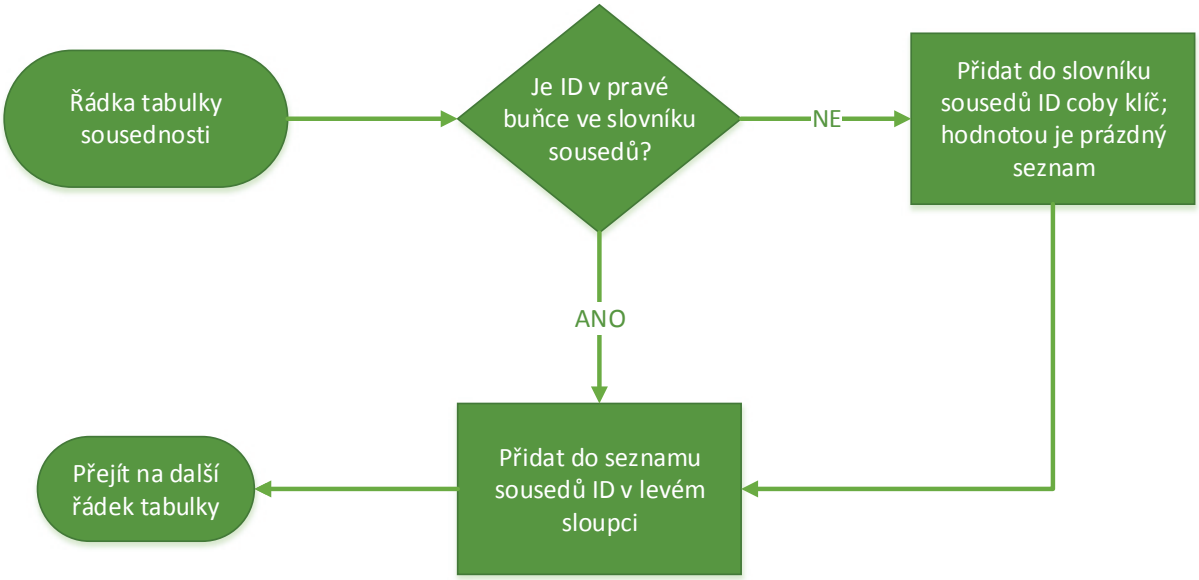

**obr. 4.1 Identifikace skupin polygonů tvořících souvislý celek**

## 4.1 Algoritmus

Pro nalezení shluků je třeba zjistit, jaké polygony spolu sousedí. K tomu byl využit nástroj *Polygon Neighbors*, který ve vstupní polygonové třídě prvků najde všechny sousednosti a uloží je do tabulky obsahující ve dvou ze svých sloupců identifikátory polygonů na obou stranách vazby sousednosti. Tato podoba však není pro následný převod do CityGML dostatečná. K převedení informace o všech sousednostech mezi dvojicemi polygonů do podoby shluků sousedících polygonů byl navržen algoritmus sestávající ze tří základních kroků:

1. Vytvoření asociativního pole předchozích sousedů
2. Přiřazení identifikátorů shluků k jednotlivým polygonům
3. Přečíslování shluků podle množiny ekvivalencí

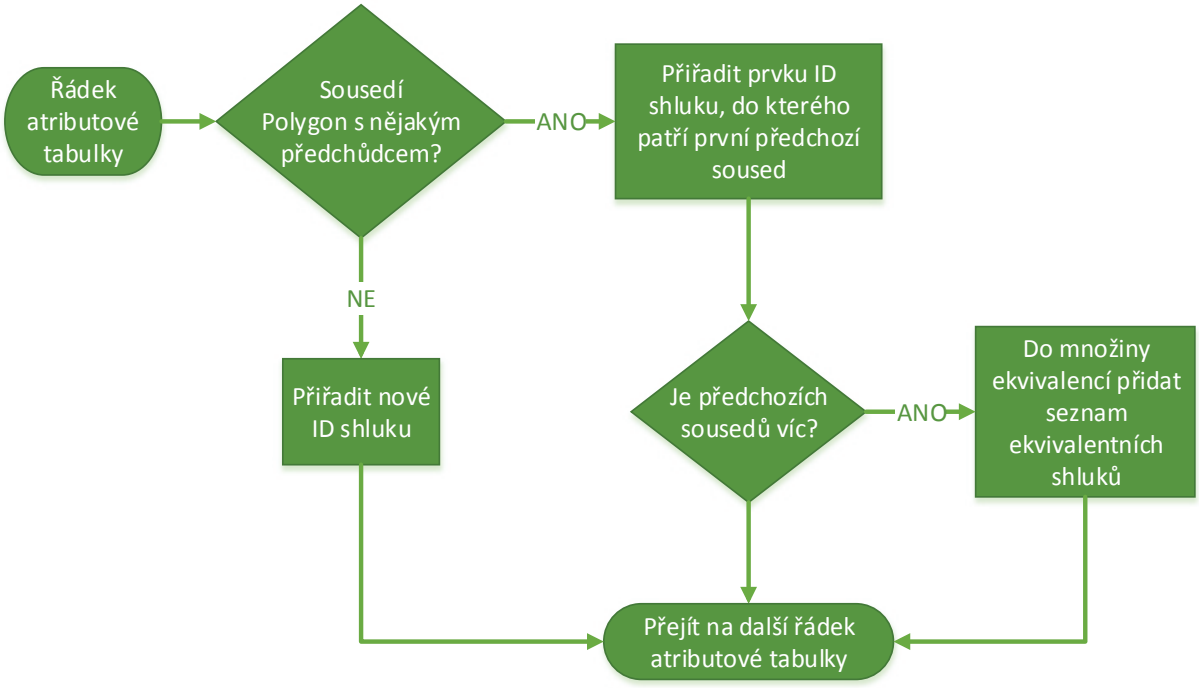
První krok vytvoří pomocnou datovou strukturu – asociativní pole obsahující pro každý polygon seznam jeho předchozích sousedů. Předchozích proto, že každou sousednost stačí znát v jednom směru (když se zaznamená, že polygon 35 má za souseda polygon 12, není třeba zaznamenávat, že polygon 12 má za souseda polygon 35). Diagram vytváření asociativního pole je na obrázku obr. 4.2.




**obr. 4.2 Diagram vytváření asociativního pole předchozích sousedů**

Stěžejní krok algoritmu je přiřazování identifikátorů shluků k polygonům. To probíhá na základě asociativního pole předchozích sousedů vytvořeného v prvním kroku. Proces přiřazování objasňuje diagram na obr. 4.3.

Takto přiřazené identifikátory však neoznačují jedinečné shluky – když spolu 2 polygony nesousedí přímo, ale přes několik sousedů, mohou být zprvu zařazeny do samostatných shluků. Jakmile se ukáže, že 2 shluky jsou ve skutečnosti spojené, označí se za ekvivalentní. Pak je třeba ještě poslední krok algoritmu, který přečísluje shluky všechny ekvivalentní shluky tak, aby byl každý shluk označen jediným identifikátorem. Z množiny ekvivalencí jsou postupně vybírány seznamy ekvivalence. Z každého z nich je vybrán identifikátor prvního shluku, kterým jsou postupně nahrazeny všechny výskyty identifikátorů ostatních shluků.




**obr. 4.3 Diagram přiřazování identifikátorů shluků jednotlivým polygonům**

## 4.2 Implementace

Navržený algoritmus byl implementován v jazyce Python. Může tak využívat balík `arcpy` zpřístupňující nástroje ArcGIS pro geoprocessing. Pro nalezení dvojic sousedících polygonů byl využit nástroj *Polygon Neighbors*, zbytek byl realizován vlastním kódem.

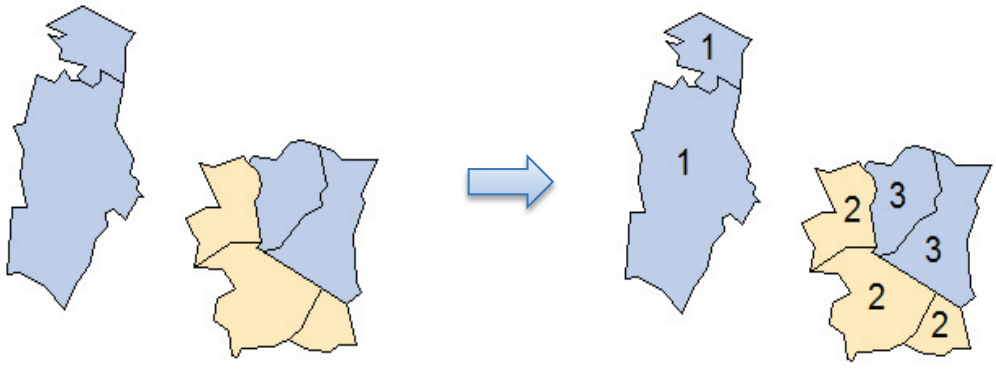
### 4.2.1 Tvorba slovníku předchozích sousedů

Pro přístup k datům byly použity kurzory z modulu data access, které prostřednictvím klauzule `with` zajišťují odstranění zámků. Následuje fragment kódu pro vytváření asociativního pole předchozích sousedů.

```
with arcpy.da.SearchCursor(tabulka,
                           ["src_" + sloupecShluku, "nbr_" + sloupecShluku])
                           as tabulkaSousednosti:
    for radka in tabulkaSousednosti:
      if (radka[1] not in sousedi):
         sousedi[radka[1]] = list() # přidá do slovníku nový seznam
      sousedi[radka[1]].append(radka[0]) # přidá src_ID do seznamu sousedů
arcpy.management.Delete(tabulka) # dále není potřeba
```

Proměnná `tabulka` odkazuje na tabulku sousednosti vytvořenou nástrojem *Polygon Neighbors*. Kurzorem jsou procházeny všechny její řádky a pro každou je přidán nový záznam do seznamu sousedů. Není-li pro příslušný polygon ve slovníku[1] `sousedi` žádný seznam sousedů, je předtím přidán nový prázdný.

Protože jsou polygony všech typů týkajících se budov (stěny, střechy, vikýře, komíny,…) uloženy ve společné třídě prvků, není možné výšeuvedené kroky provádět pro všechny polygony této třídy najednou. Výsledkem by totiž byly shluky odpovídající blokům budov, protože v rámci bloku tvoří všechny polygony souvislý celek sousedů. Hledání dvojic sousedních polygonů a následné plnění slovníku předchozích sousedů je tedy prováděno postupně pro výběry polygonů odpovídající jednotlivým typům. Výsledné shluky pak respektují rozdílné typy prvků, jak ilustruje obr. 4.4.




**obr. 4.4 Identifikace skupin polygonů různých typů**

### 4.2.2 Přiřazování identifikátorů shluků jednotlivým polygonům

V druhém kroku algoritmu byl opět využit `SearchCursor`, přičemž tentokrát je procházena třída prvků obsahující polygony, jejichž shluky program hledá. Program

[1] Pro asociativní pole, v některých programovacích jazycích nazývané mapa, se v Pythonu používá pojem slovník.

nahlížením do slovníku `sousedi` zjišťuje, zda polygon sousedí s některým již zpracovaným. Pokud ano, přiřadí mu jeho identifikátor shluku. Jinak mu přiřadí identifikátor shluku nový, dosud nepoužitý. Zjednodušený[2] fragment kódu pro přiřazování identifikátorů shluků polygonům je uveden níže.

```
with arcpy.da.SearchCursor(tridaPrvku, [sloupecID]) as trida:
   for prvek in trida:
      if (prvek[0] in sousedi): # sousedí s aspoň jedním předchozím prvkem
         sousediPrvku = sousedi[prvek[0]]
         shluky[prvek[0]] = shluky[sousediPrvku[0]]
         if (len(sousediPrvku) > 1): # prvek má víc sousedů
            mnozinaEkvivalence = set()
            for soused in sousediPrvku:
               mnozinaEkvivalence.add(shluky[soused])
            if (len(mnozinaEkvivalence) > 1):
               ekvivalentniShluky.add(tuple(sorted(mnozinaEkvivalence)))
      else:
         if (len(shluky) == 0):
            shluky[prvek[0]] = 1 # v případě prázdného slovníku shluků
         else:
            shluky[prvek[0]] = max(shluky.values()) + 1
```

V situaci, kdy má nějaký polygon předchozích sousedů více, je vytvořena množina ekvivalence, do které jsou všechny tyto přidány. Datová struktura množina je použita proto, aby v případě více sousedů patřících do stejného shluku (k takovým situacím dochází velice často) nevznikaly zbytečné záznamy o ekvivalenci stejných shluků. K přidání množiny ekvivalence (převedené na n-tici) do množiny ekvivalencí se přistupuje jen tehdy, obsahuje-li více než 1 záznam. Tím je snížen počet ekvivalencí, což urychlí poslední krok algoritmu.

### 4.2.3 Přečíslování shluků podle množiny ekvivalencí

Za účelem přečíslování shluků je množina ekvivalencí převedena na seznam, který je procházen. Z každé n-tice ekvivalence je vybrán první shluk. Výskyt všech ostatních je pak hledán ve slovníku shluků, kde je nahrazován právě tím prvním. Následuje fragment kódu zjednodušen obdobně jako v předchozím případě.

```
for enticeEkvivalence in seznamEkvivalentnichShluku:
   prvniShluk = 0 # první shluk z entice ekvivalence
   for shlukEkvivalence in enticeEkvivalence:
      if prvniShluk == 0: # první shluk je třeba vždy vynechat z iterace
         prvniShluk = shlukEkvivalence
      else: # pro všechny shluky z entice kromě prvního
         for (prvek, shlukPrvku) in shluky.iteritems():
            if (shlukPrvku == shlukEkvivalence):
               shluky[prvek] = prvniShluk
```

Proměnná `seznamEkvivalentnichShluku` značí seznam vytvořený z množiny `ekvivalentniShluky` plněné v předchozím kroku algoritmu za účelem jeho iterování. Následuje už jen zápis identifikátorů shluků do nově vytvářeného sloupce atributové tabulky třídy prvků s polygony. Kompletní skript se nachází v příloze.

[2] Příkazy pro manipulaci s uživatelským rozhraním jsou vynechány.

### 4.2.4 Výpočetní náročnost

Složitost vlastního hledání sousedů je $O(n^2)$, protože se testuje sousednost každého polygonu s každým. Přiřazování shluků jednotlivým polygonům má složitost O(n). Doba běhu obou kroků je v zásadě úměrná počtu zpracovávaných polygonů, přičemž druhý krok trvá déle pro data s více sousednostmi.

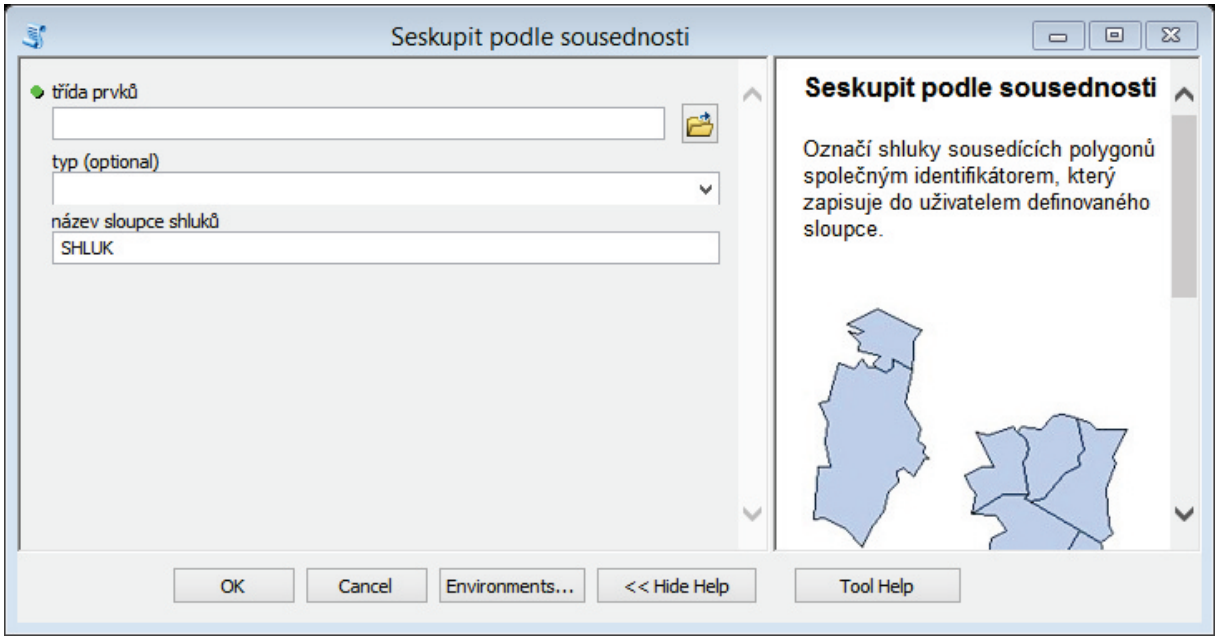
Problematický je odhad časové náročnosti přečíslování shluků podle množiny ekvivalencí. Velikost této množiny totiž silně závisí na charakteru vstupních dat. Pro třídy prvků obsahující ostrovy polygonů (to jsou v této práci například komíny či vikýře) je množin ekvivalencí poměrně málo a přečíslování shluků tak trvá oproti hledání sousedů kratší dobu. Naopak pro data obsahující velké skupiny sousedících polygonů (v této práci například stěny budov) vznikne při vytváření shluků takové množství ekvivalencí, že jejich následné přečíslování je časově nejnáročnější operací celého algoritmu. Doba běhu vstupních a výstupních operací je zanedbatelná, všechny pomocné datové struktury včetně prvkových tříd jsou ukládány pouze do operační paměti, k zápisu na disk tak dochází až na konci skriptu.

### 4.2.5 Vývoj

Součástí instalace Pythonu je IDLE, vývojové prostředí pro Python. To však disponuje pouze základní funkcionalitou, chybí dokonce i našeptávání jmen proměnných a funkcí. Proto bylo použito vývojové prostředí PyDev. Zvoleno bylo proto, že se jedná o doplněk do prostředí Eclipse.

PyDev se instaluje standardním způsobem přímo z rozhraní této aplikace. Po instalaci je však třeba jej nastavit, a to zejména cestu k interpretu jazyka Python a ke knihovnám ArcGIS. To umožní ladění skriptů využívajících balík `arcpy` přímo z Eclipse. Potom už je možné založit nový Python projekt, přidat do něj nový balík a vytvořit první modul. Ten je pak možné použít jako zdrojový soubor nástroje pro geoprocessing vytvářeného v prostředí ArcGIS. Celý proces instalace a nastavování PyDev je popsán v [4].

## 4.3 Nástroj pro geoprocessing

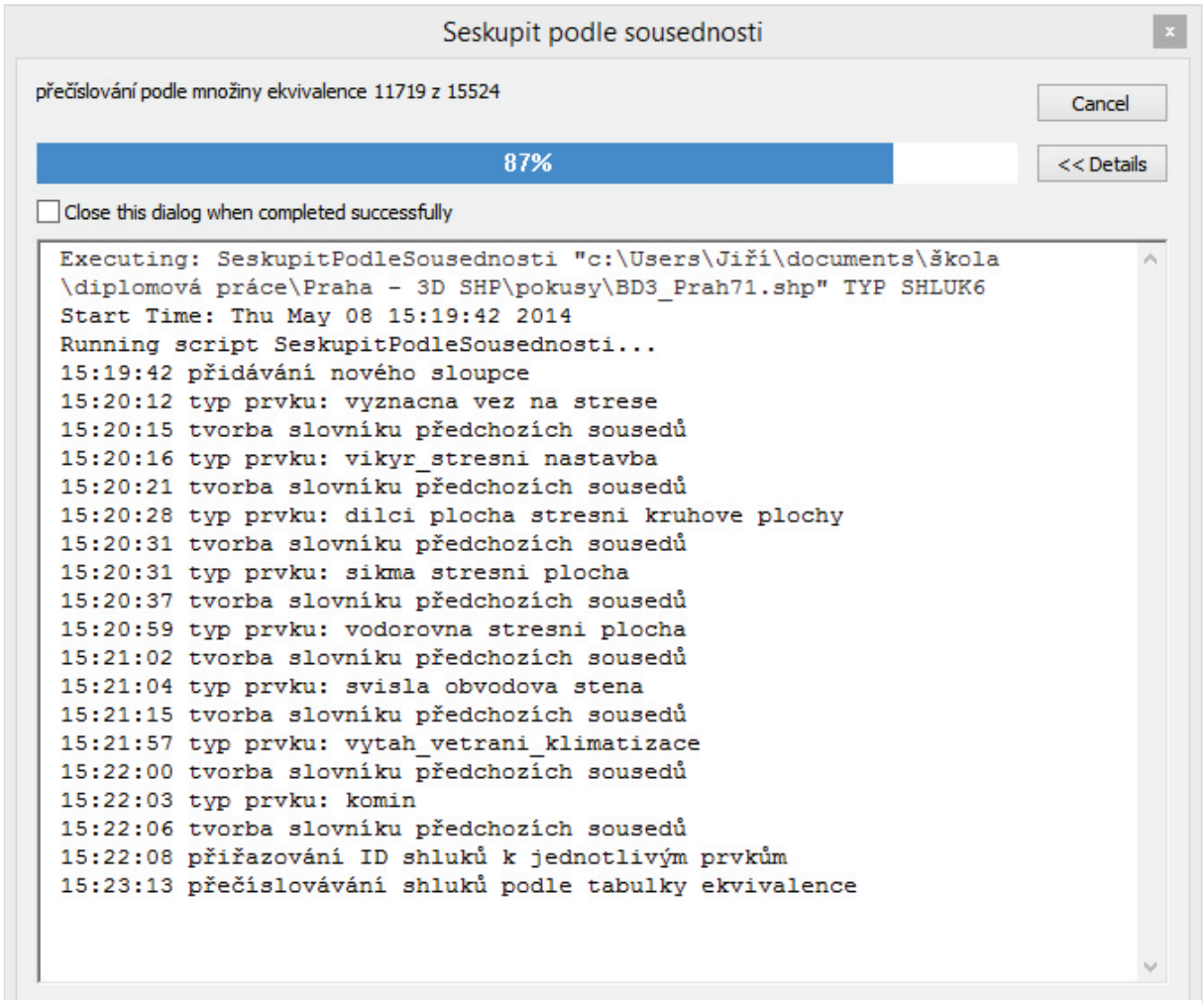



**obr. 4.5 Dialogové okno nástroje *Seskupit podle sousednosti***

Pro umožnění pohodlného spouštění skriptu z prostředí ArcGIS byl vytvořen nástroj *Seskupit podle sousednosti*. Jeho povinným vstupním parametrem je třída prvků, jejíž polygony budou seskupeny do shluků podle jejich sousednosti, a také název nového sloupce, do kterého budou zapsány identifikátory těchto shluků. Dialogové okno je na obr. 4.5.

Uživatel nastavuje 2 povinné parametry a 1 volitelný.

- třída prvků
  - prvková třída, jejíž polygony budou seskupeny do shluků podle jejich sousednosti
- typ (volitelný parametr)
  - sloupec atributové tabulky obsahující typy (vrstvy, druhy) prvků
  - je-li použit, provede nástroj seskupování polygonů podle sousednosti zvlášť pro polygony s každou jedinečnou hodnotou tohoto atributu
  - budou-li ve třídě prvků 2 polygony s různou hodnotou tohoto atributu sousedit, budou přiřazeny do různých shluků.
- název sloupce shluků
  - název pro nový sloupec atributové tabulky, který bude obsahovat identifikátory shluků
  - smí obsahovat pouze alfanumerické znaky.
  - nesmí být delší než 10 znaků
  - nesmí se shodovat s názvem sloupce, který již v dané třídě prvků existuje




**obr. 4.6 Běžící nástroj *Seskupit podle sousednosti***

Součástí vytvořeného nástroje je i validace vstupních parametrů, která zajistí vhodnou volbu názvu nového sloupce. Zahrnuje omezení délky názvu a použitých znaků a také

kontrolu existence sloupce ve třídě prvků. Zajištěna je také aktualizace schématu výstupní třídy prvků po každé změně hodnoty parametru název sloupce shluků, což umožňuje použití nástroje v modelech.

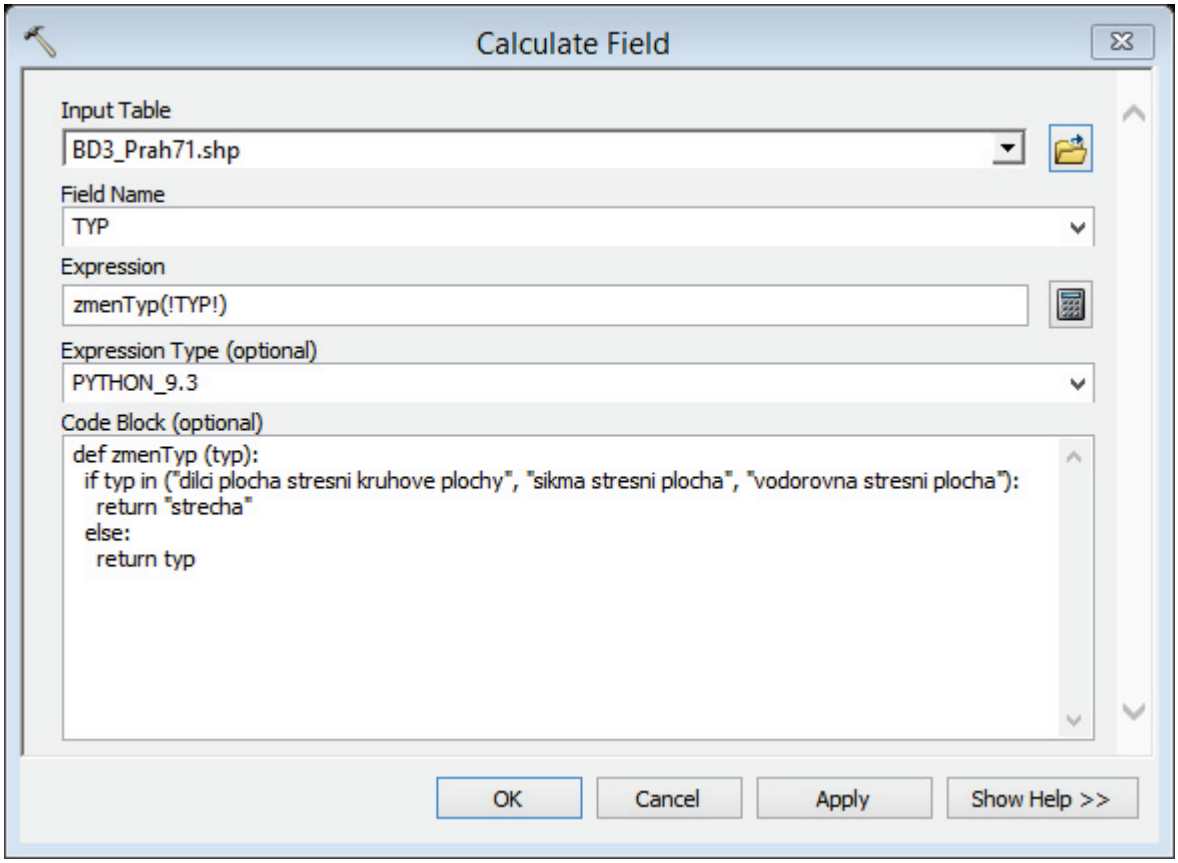
Protože doba běhu nástroje může být nezanedbatelná, bylo implementováno i předávání zpráv o aktuální činnosti do prostředí ArcGIS a ovládání ukazatele průběhu. Ten je však pouze orientační, neboť nejen celková doba běhu, ale i poměr trvání jednotlivých kroků algoritmu je silně závislý na charakteru vstupních dat. Okno spuštěného nástroje je na obr. 4.6. K nástroji je připravena i základní nápověda včetně vysvětlujících ilustrací.

Zpracování třídy prvků budov s výřezem z Digitálního modelu zástavby a zeleně hlavního města Prahy, který obsahuje cca 160 000 polygonů, trvá na počítači s procesorem Intel Core 2 Duo 2,66 GHz 11 minut a 20 sekund.

## 4.4 Využití pro identifikaci prvků na budovách

Vytvořený nástroj lze úspěšně použít pro identifikaci objektů na budovách, jako jsou komíny a vikýře. Problém nastává v případě střech. Ty jsou totiž v digitálním modelu zástavby a zeleně rozlišeny na 3 různé skupiny – šikmá střešní plocha, vodorovná střešní plocha a dílčí plocha střešní kruhové plochy. Nástroj *Seskupit podle sousednosti* vytvoří shluky s ohledem na toto rozdělení. V případě přítomnosti různých typů střechy na 1 budově tak střechu rozdělí na více objektů, což pro následný převod do CityGML není účelné.

Aby k rozdělování střech na více objektů nedocházelo, je před spuštěním nástroje nutné sloučit všechny druhy střech do jedné třídy (např. střechy). K tomu lze využít nástroj *Calculate Field* s použitím pole kódu, jak je ukázáno na obr. 4.7. Všechny druhy střechy budou nahrazeny typem „střecha“ při současném zachování ostatních hodnot ve sloupci TYP.




**obr. 4.7 Sjednocení typů střech nástrojem *Calculate Field***

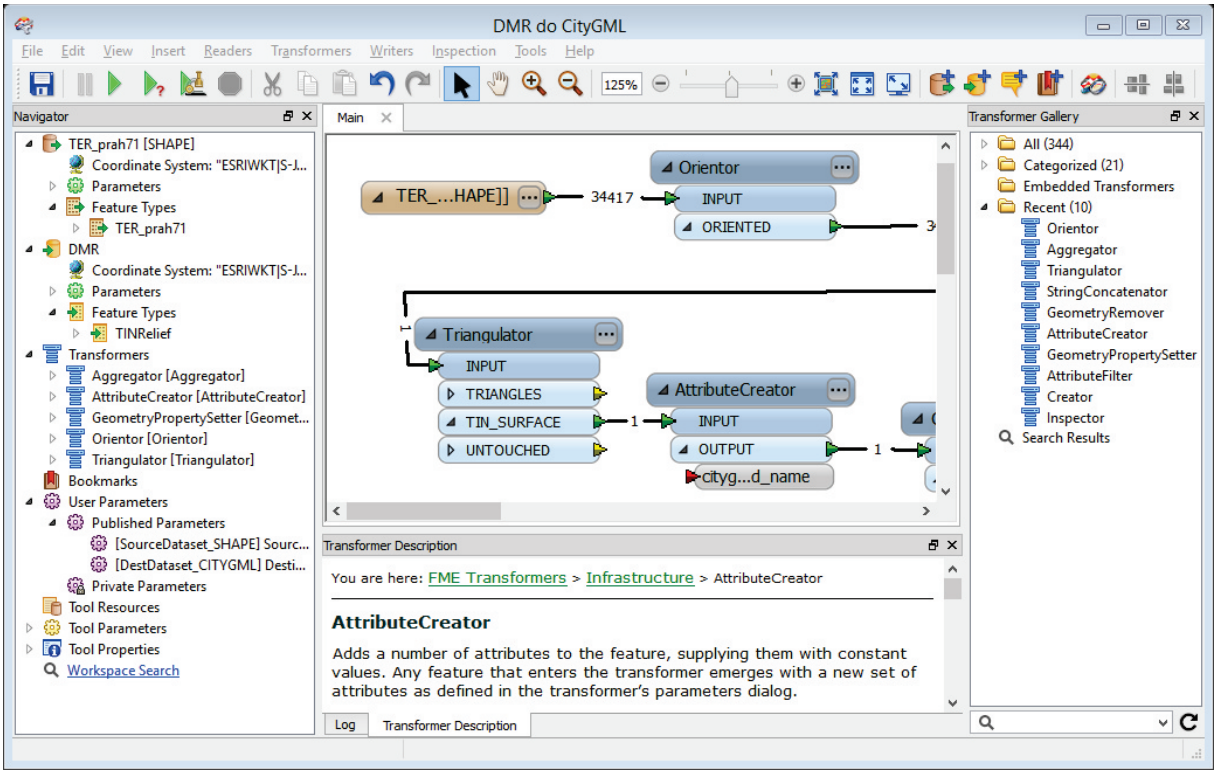
# 5 Převod do CityGML

V této kapitole je popsán převod Digitálního modelu zástavby a zeleně z formátu Shapefile do CityGML pomocí ArcGIS Desktop. Samotný ArcGIS export dat do CityGML neumožňuje, podporu tohoto formátu přidává až nadstavba Data Interoperability. Ta umožňuje z prostředí ArcGIS manipulovat s prostorovými daty v mnoha různých formátech. Jedná se o omezenou verzi aplikace FME (Feature Manipulation Engine) integrovanou do prostředí ArcGIS.

Kromě prostého importu a exportu dat mezi formáty používanými v ArcGIS a jinými formáty prostorových dat umožnuje i vytvoření vlastního nástroje pro pokročilou transformaci prostorových dat (tzv. Spatial ETL[3] Tool). Právě to je potřeba při exportu dat do CityGML, poněvadž mapování GIS atributů na CityGML třídy je poměrně složité a musí být definováno vždy pro konkrétní datový model. Návrh nástroje probíhá v aplikaci FME Workbench. Vytvořený nástroj je pak možné spouštět z prostředí ArcGIS samostatně, přidat ho do modelu tvořeného v ModelBuilderu, nebo jej volat ze skriptů psaných v Pythonu. [5]

## 5.1 Ovládání FME Workbench

Pro vytvoření transformačního nástroje je třeba například v aplikaci ArcCatalog v kontextové nabídce nějakého předem vytvořeného (nikoliv systémového) toolboxu zvolit možnost *new → Spatial ETL Tool*. Pak stačí jen v průvodci vybrat zdrojový (v případě této práce ESRI Shape) a cílový (CityGML) formát ze široké nabídky, zvolit umístění zdrojových dat a nastavit verzi CityGML.




**obr. 5.1 FME Workbench**

[3] ETL (Extras, Transform, Load) označuje proces získávání, zpracování a ukládání dat; ETL nástroj tedy zvládne čtení dat, jejich převod do jiného formátu (včetně případného filtrování, slučování atributů a generování nových) a zápis výsledných dat

Po dokončení průvodce se zobrazí okno aplikace FME Workbench. Tam probíhá návrh nástroje pro převod dat, který se skládá z jednotlivých komponent pro čtení, transformaci a zápis dat a jejich spojení. Protože návrh probíhá graficky, vzniká kromě vlastního nástroje i jeho přehledné schéma. K orientaci v následujícím textu poslouží obr. 5.1 s hlavním oknem aplikace. Z důvodu problémů při ukládání nástroje v situacích, kdy ArcGIS aplikace, ze které byl nástroj otevřen, přestane odpovídat (k tomu dochází při práci s velkým objemem dat velice často), byla pro snadnější návrh nástroje pro převod budov použita zkušební verze FME Desktop. Ovládání aplikace FME Workbench je však v obou případech stejné, následující popis tak pokrývá obě možnosti.

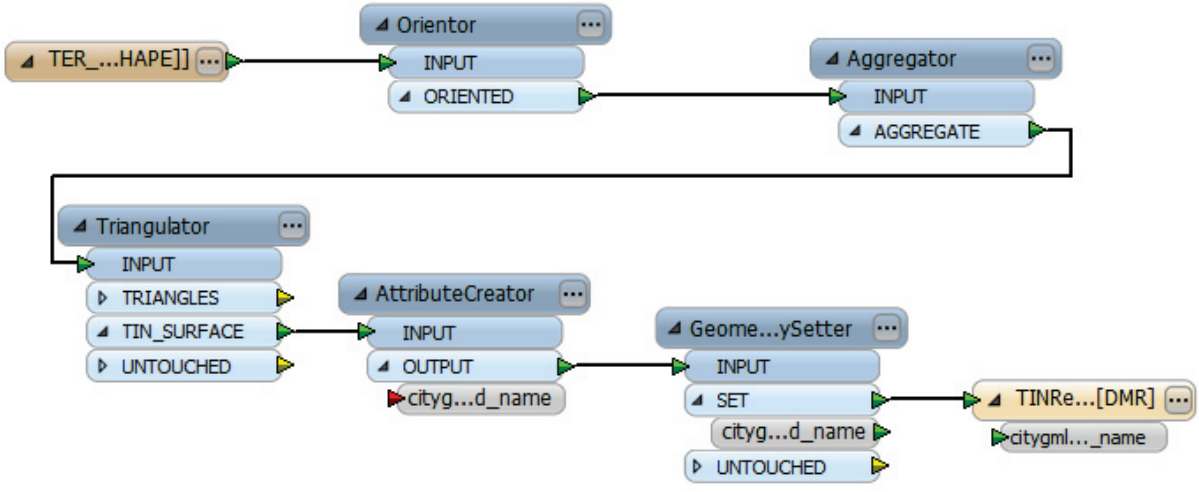
V horní části levého sloupce okna (Navigator) lze změnit průvodcem nastavené vlastnosti pro čtení a zápis, jako umístění souborů, souřadnicový systém a parametry specifické pro daný datový formát. Stěžejní část práce s aplikací probíhá v centrální části okna, na tzv. plátně (na obr. 5.1 uprostřed). Příprava transformačního nástroje spočívá v nastavení vstupních a výstupních tříd prvků (na plátně podbarvené hnědě), přidávání transformačních komponent (na plátně podbarvené modře), jejich pospojování a nastavení. Ovládání aplikace FME Workbench je detailně popsáno v [6].

U vstupních tříd je třeba zvolit atributy, které budou vstupovat do procesu transformace. U výstupních pak zejména atributy formátu. Pro zápis geometrie do CityGML je klíčový atribut `citygml_lod_name`, který definuje vazbu na geometrii. U tříd lze nastavit pouze přítomnost nebo nepřítomnost jednotlivých atributů, jejich hodnoty je třeba nastavit použitím některé transformační komponenty.

Transformační komponenty slouží k provádění rozličných operací s daty. Může se jednat o změny geometrie (např. generalizace, topologická překrytí, filtrování na základě prostorových dotazů,…) nebo atributů (změny jmen či hodnot stávajících atributů, vytváření nových, filtrování na základě atributů,…). V aplikaci je k dispozici velké množství těchto komponent. Jsou roztříděny do skupin a zobrazeny v tzv. galerii (na obr. 5.1 vpravo). Jejich popis je v [7].

## 5.2 Převod digitálního modelu reliéfu

Pro převod DMR do CityGML je potřeba série pěti transformačních komponent. Nejprve je nutné nastavit orientaci polygonů. Zatímco v SHP na ní nezáleží (polygony jsou zobrazovány oboustranně), CityGML povrchy polygonů rozlišuje. K nastavení poslouží komponenta `Orientor`, která jednotlivým trojúhelníkům nastaví takovou orientaci, aby byly při vizualizaci zobrazovány při pohledu shora.




**obr. 5.2 Schéma převodu DMR do CityGML**

Dalším krokem je sloučení všech trojúhelníků do jednoho objektu, které provede komponenta `Aggregator`. Bez toho by na výstupu bylo množství objektů třídy `gml:TrianguledSurface`, z nichž každý by obsahoval jediný trojúhelník. Parametr *Group By* zůstane nevyplněn, protože cílem je dostat všechny prvky do jedné skupiny. Pro DMR nejsou potřeba žádné atributy, další nastavení komponenty tak není podstatné.

Dále je třeba převést množinu trojúhelníků na TIN. Tuto operaci provede komponenta `Triangulator`, která rozděluje plochy na trojúhelníky. Protože všechny plochy už jsou trojúhelníkové, k žádnému zásahu do geometrie nedojde. Pouze se nastaví správný typ geometrie.

Přestože pro geometrii typu `gml:TrianguledSurface` je v CityGML přípustná pouze jediná vazba, a to `dem:tin`, je třeba tuto vazbu geometrii nastavit. Stará se o to atribut `citygml_lod_name` zmíněný v kapitole 5.1. Navzdory svému jménu se jeho pomocí nenastavuje jen LOD, ale všechny vazby na geometrii v datovém modelu CityGML. Atribut `citygml_lod_name` je vytvořen pomocí komponenty `AttributeCreator`, jeho hodnota je nastavena na `tin`.

Zbývá nastavit nově vytvořený atribut coby typ geometrie. O to se postará komponenta `GeometryPropertySetter`, pro kterou bude zdrojovým atributem právě vytvořený `citygml_lod_name`. Tím je již vše správně nastaveno a veškerá geometrie bude zapsána do jediné třídy prvků, a to `TINRelief`. Schéma celého procesu je na obr. 5.2. Nástroj je součástí toolboxu vytvořeného v rámci této práce. Nachází se na přiloženém CD.

## 5.3 Převod budov

Zatímco v případě DMR byl celý proces lineární, převod budov je o poznání složitější. Je potřeba vyřešit několik zásadních problému zmiňovaných v kapitole 0. Jednotlivé kroky převodu jsou popsány v následujících podkapitolách. Součástí popisu jsou i výřezy ze schématu převodního nástroje. Kompletní schéma je pro svou obsáhlost uvedeno v příloze 3. Obsahuje celkem 5 výstupních typů (budovy, prvky na budovách, stěny, střechy a spodní části budov) a 44 transformačních komponent. Čísla u spojovacích linií ukazují počet prvků, které danou spojnicí prošly při posledním běhu nástroje, tj. při zpracování všech polygonů z jednoho SHP, kterých je něco přes 160 tisíc.

První komponenta, `GeometryFilter`, propustí dále pouze polygony. Je zařazena proto, že ve vstupních datech byl zjištěn prvek bez geometrie, což působilo v dalších částech procesu problémy. Z obrázku v příloze 3 je vidět, že tímto filtrem neprošel pouze jeden prvek. Za něj je zařazen `AttributeCreator`, který před identifikátory budov předřadí řetězec „budova“ a před identifikátory shluků řetězec „shluk“. To je nutné z důvodu použití těchto atributů jako XML atributů `gml:id` ve výstupním GML souboru. Tento atribut je typu `xs:ID`, jeho hodnoty tak smějí začínat pouze písmenem nebo podtržítkem [8].

### 5.3.1 Nastavení orientace polygonů

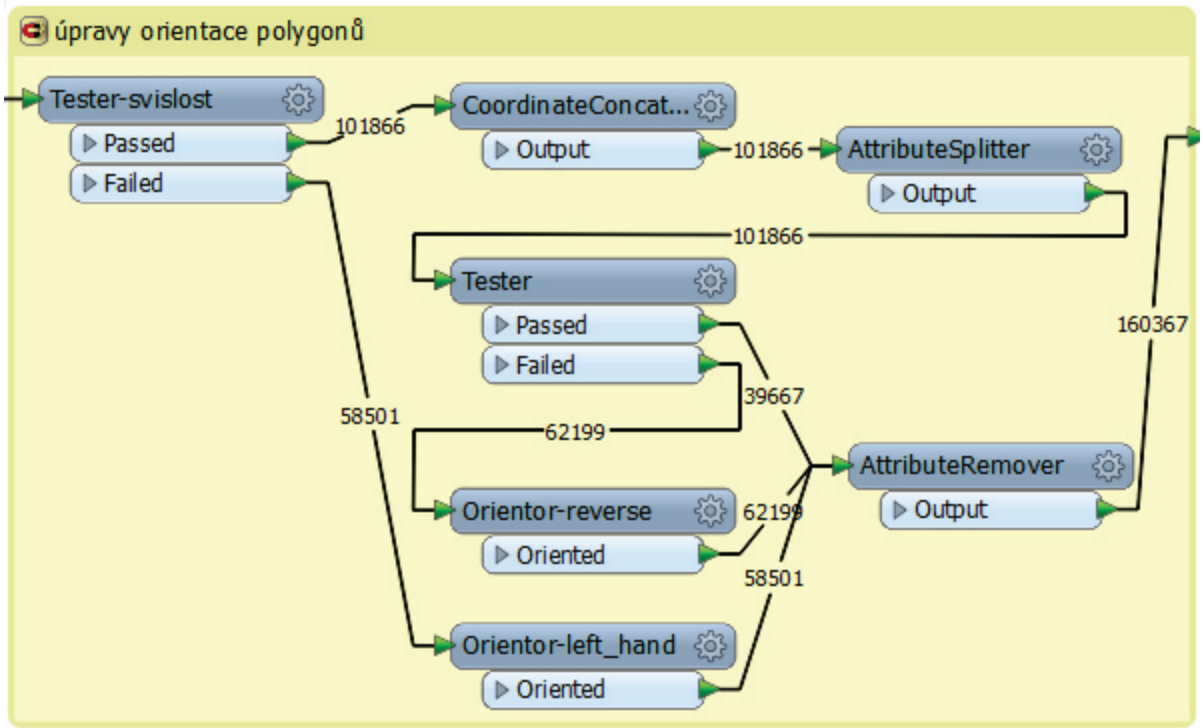
Stejně jako v případě DMR je třeba nastavit všem polygonům správnou orientaci. Nyní je však tento úkol složitější. Nesvislé polygony je možné orientovat proti směru hodinových ručiček, čímž budou viditelné shora. Svislé polygony však takovým způsobem nastavit nelze. Protože polygony tvoří povrchy jednotlivých objektů (např. budov), jejichž vnitřky nejsou modelovány, je třeba nastavit polygonům takovou orientaci, aby byly viditelné z vnějšku jednotlivých objektů.

V ideálním případě by bylo možné správnou orientaci polygonů nastavit například podle směru přilehlé střechy (v případě stěn budov) či jiného nesvislého polygonu (v případě komínů apod.). Stačilo by pro každý svislý polygon nalézt jeden nesvislý, který s ním po průmětu do vodorovné roviny sousedí, a určit polohu jeho geometrického středu vzhledem k nejspodnější (orientované) úsečce z hranice polygonu. Pokud by se nacházel vlevo, znamenalo by to, že polygon je orientován správně. Pro takové testování však není aplikace FME Workbench ideálním nástrojem, proto bylo rozhodování o správnosti orientace polygonu zjednodušeno tak, aby k němu nebylo potřeba jiného než právě testovaného polygonu.

V datech Digitálního modelu zástavby a zeleně hlavního města Prahy byla vypozorována určitá pravidelnost ve volbě počátečního bodu polygonu. Tento se pro většinu objektů nachází v pravém spodním rohu při pohledu z vnějšku objektu. Orientace polygonů však již jednotná není. Pro přibližně polovinu polygonů je druhým bodem bod nad počátkem (liší se souřadnice Z), pro zbytek je to bod při zemi (stejná souřadnice Z, liší se X a Y). Zatímco v prvním případě se polygony po převodu do CityGML zobrazují správně (viditelné z vnějšku objektů), v případě druhém je třeba orientaci polygonů otočit.

Výše zmíněná závislost byla vypozorována pro polygony s více než třemi vrcholy, u trojúhelníků se nejčastější umístění počátku jevilo jako levý spodní roh. Jelikož opět platí, že orientace polygonu musí být proti směru hodinových ručiček při pohledu z vnějšku objektu, je pro rozhodnutí o tom, zda orientaci daného polygonu otočit či nikoliv, testován rozdíl výšek druhého a třetího vrcholu. Orientace polygonu se otáčí v případě, že je třetí vrchol níž než druhý.

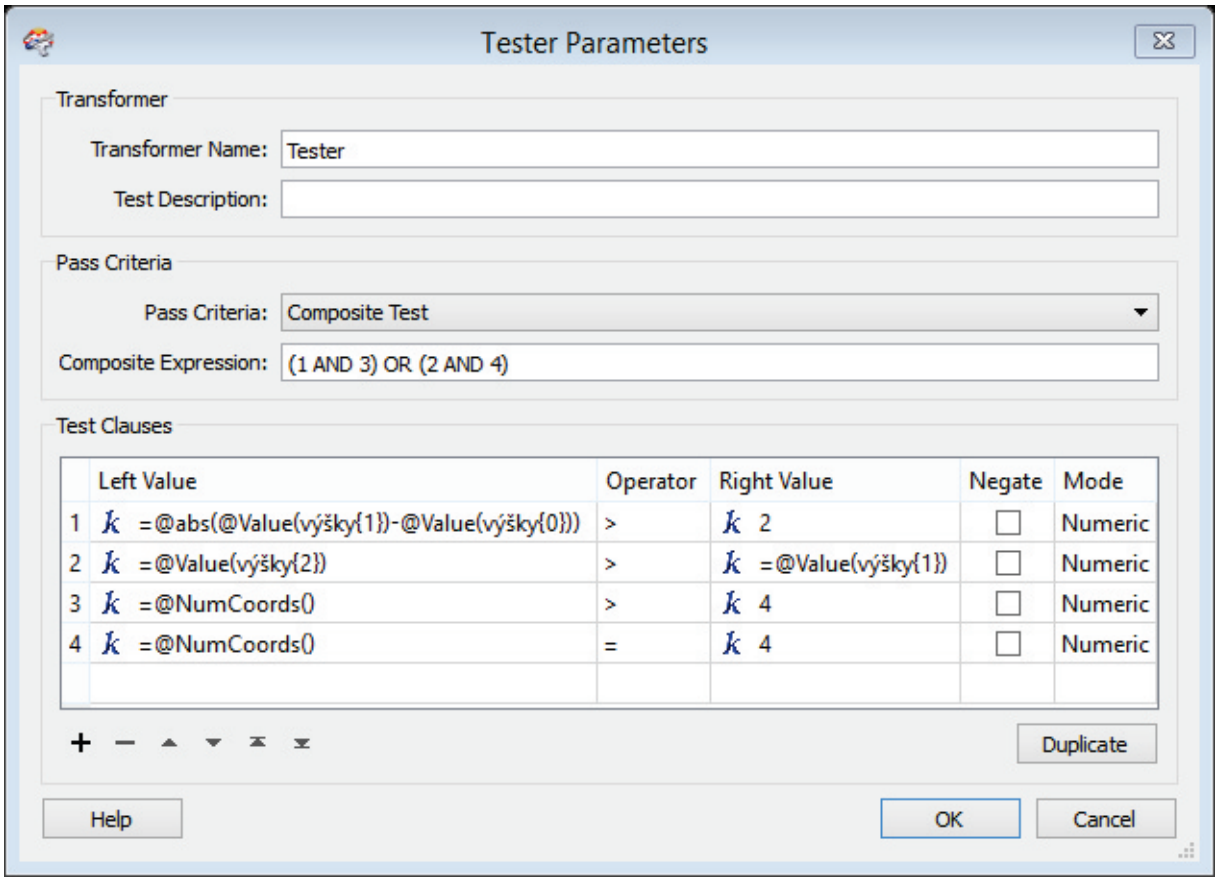
Přestože byl pro zjišťování orientace svislých polygonů používán takto zjednodušený postup, je výsledek pro většinu polygonů správný. Schéma nastavování orientace polygonů je na obr. 5.3. Čísla u spojnic značí opět počet prvků, který danou spojnicí prošel při posledním běhu nástroje.




**obr. 5.3 Schéma nastavování orientace polygonů**

Úvodní `Tester` propouští portem *Passed* svislé polygony. Testovaná je hodnota plochy, která musí být nulová (výpočet plochy probíhá ve 2D). Ostatní polygony jsou posílány na port *Failed*. Pro ty je nastavení orientace jednoduché, stejně jako v případě DMR je nastavena proti směru hodinových ručiček (`left_hand`).

Pro určení orientace svislých polygonů podle výše zmíněného postupu je třeba znát výšky jeho jednotlivých vrcholů. Ty z geometrie získá a do atributu uloží komponenta `CoordinateConcatenator`. Následný `AttributeSplitter` tento atribut převede na seznam, k jehož jednotlivým prvkům již má přístup Tester. Ten pustí portem *Passed* správně orientované polygony a portem *Failed* polygony, jejichž orientaci je třeba obrátit. O to se již postará `Orientor` s volbou reverse. Konečný `AttributeRemover` pouze odstraní atributy, které již nejsou dále potřeba, aby byly další části schématu přehlednější.



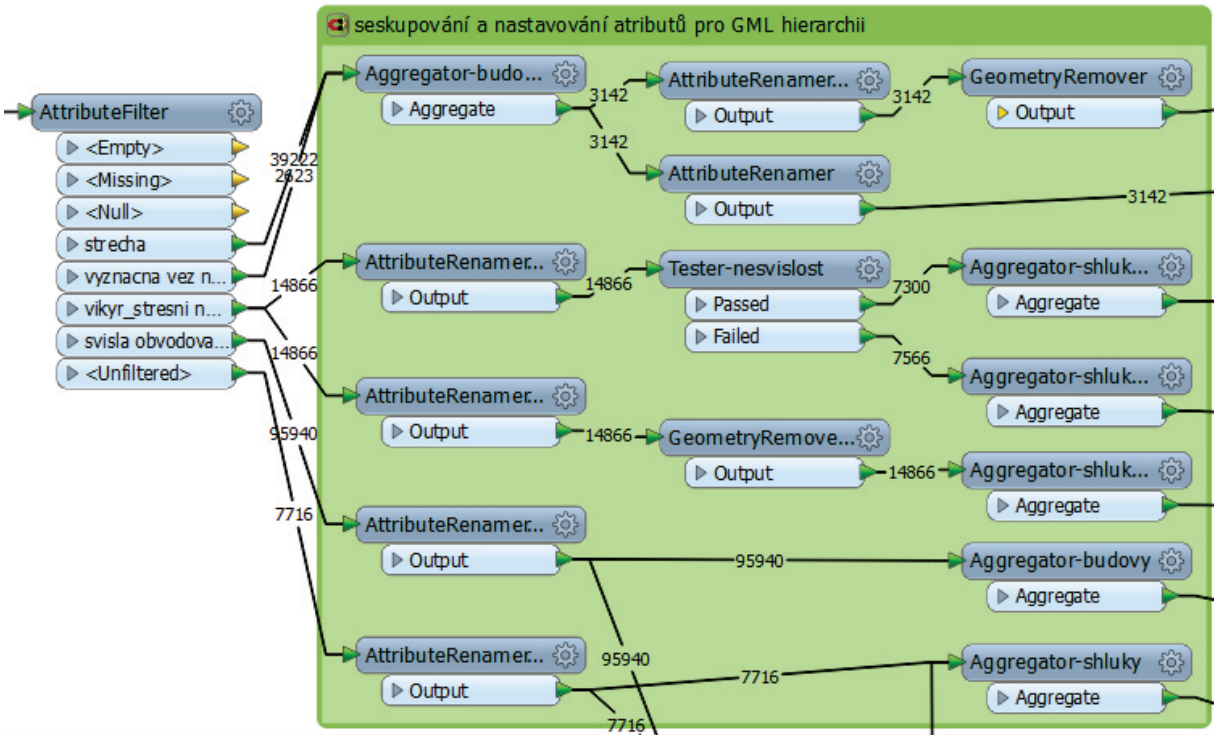

**obr. 5.4 Testovací podmínka pro určení orientace polygonů**

Testovaná podmínka je vidět na obr. 5.4. Je složená ze 4 částí, přičemž musí být splněny buď liché, nebo sudé řádky, jak definuje položka *Composite Expression*. Tento způsob byl zvolen proto, aby mohly být najednou testovány podmínky pro trojúhelníky a pro ostatní polygony. Ty jsou rozlišeny spodními dvěma řádky. Porovnávaná hodnota je 4, protože ve FME jsou polygony uloženy tak, že počáteční a koncové body jejich hranic mají vždy stejné souřadnice (bod je tedy uložen duplicitně).

Druhý řádek podmínky testuje, zda je souřadnice Z třetího vrcholu trojúhelníka větší než souřadnice Z jeho druhého vrcholu. Souřadnice Z se nacházejí v seznamu *výšky* vytvořeném komponentou `AttributeSplitter`. Seznamy jsou ve FME indexovány od 0. První řádek podmínky pak testuje, jestli se výšky prvních 2 vrcholů polygonu dostatečně liší.

### 5.3.2 Seskupování polygonů do objektů a budování jejich hierarchie

Klíčovou částí tvorby dat v CityGML je vhodné vytváření objektů a jejich hierarchického uspořádání. To musí odpovídat specifikaci [1], v případě budov pak modelu na obr. 3.4, kde jsou již zvýrazněné třídy, které se budou ve výsledném souboru vyskytovat, a agregační vazby mezi nimi. Schéma pro tuto část transformace je na obr. 5.5.




**obr. 5.5 Schéma seskupování polygonů do objektů a budování jejich hierarchie**

Ústřední komponentou celého procesu je `Filter`, který polygony třídí podle toho, do jaké třídy patří, a posílá je na příslušné porty. O seskupování polygonů do shluků se postará `Aggregator`. Vytváří kolekce prvků na základě společné hodnoty zvoleného atributu. Polygony jsou do kolekcí seskupovány podle identifikátoru budovy (platí pro stěny a střechy) nebo podle identifikátoru shluku, o jehož tvorbě pojednává kapitola 4 (platí pro ostatní prvky).

Tyto identifikátory neslouží jen k seskupování do objektů, ale i k realizaci agregačních vazeb. Ty jsou ve FME definovány prostřednictvím atributů formátu CityGML – `gml_id` a `gml_parent_id`. Jejich hodnoty nastavuje `AttributeRenamer`. Nevytváří nové hodnoty, ale přejmenovává existující atributy (identifikátory budov a shluků) na tyto atributy formátu.

Hodnota atributu `gml_id` se ve výsledném GML souboru projeví v hodnotě atributu `gml:id` příslušného XML elementu. Tento atribut je povinný, atribut `gml_id` však není nutné specifikovat. Není-li definován uživatelem, vygeneruje hodnotu pro `gml:id` FME [9]. Atribut `gml_id` je třeba použít v případě, že danému elementu mají být přidáni potomci. Každý prvek, který má být ve výstupním GML souboru potomkem nějakého XML elementu, totiž musí mít nastavený atribut `gml_parent_id`, a to na hodnotu rovnou hodnotě `gml_id` rodiče.

Z obr. 5.5 je patrné, že polygony střech a význačných věží na střechách jsou zpracovávány společně. To proto, že jako význačné věže na střechách jsou v Digitálním modelu zástavby a zeleně hlavního města Prahy označeny pouze polygony střech těchto věží; jejich zdi jsou

modelovány jako svislé obvodové stěny. Aggregator polygony střech seskupí do kolekcí, které budou ve výstupním GML souboru reprezentovány elementem gml:MultiSurface.

Dále se cesta ve schématu dělí. V těchto případech pošle FME každý příchozí prvek oběma větvemi. Tomu odpovídá i počet znázorněný u obou čar, který říká, kolik kolekcí bylo vytvořeno z celkových 41854 prvků vstupujících do komponenty Aggregator. AttributeRenamer v horní větvi přejmenuje na gml_id atribut uchovávající identifikátor budovy. Po následném odstranění geometrie komponentou GeometryRemover tak budou vytvořeny rodičovské elementy pro jednotlivé budovy. Ty pak budou obsahovat všechny kolekce polygonů tvořící střechy, stěny a všechny další prvky, protože každý polygon náleží nějaké budově. Střechy vzniknou z kolekcí procházejících spodní větví, kde AttributeRenamer označí identifikátory budov jako gml_parent_id.

Zpracování vikýřů a střešních nadstaveb se větví hned v počátku. Spodní cesta slouží k vytvoření rodičovských elementů bldg:BuildingInstallation, jejichž potomky budou všechny polygony vikýřů uspořádané do kolekcí. Aby se na něj tyto polygony mohly odkazovat, přejmenuje AttributeRenamer atribut uchovávající identifikátor shluku na gml_id. Všechny prvky se stejnou hodnotou identifikátoru shluku budou patřit do jednoho objektu bldg:BuildingInstallation. Jelikož každý takový objekt náleží nějaké budově, nastaví AttributeRenamer identifikátor budovy jako gml_parent_id. Protože geometrie bude uložena až u jednotlivých typů povrchu, následuje GeometryRemover. Konečný Aggregetor zabrání vytváření duplicitních elementů bldg:BuildingInstallation.

AttributeRenamer v horní větvi Zpracování vikýřů a střešních nadstaveb přejmenuje na gml_parent_id právě výše zmíněnou hodnotu identifikátoru shluku. Následný Tester pošle na port *Passed* nesvislé polygony a na port *Failed* ty svislé. Oboje budou komponentou Aggregator seskupeny podle hodnoty atributu gml_parent_id, což je původní identifikátor shluku, tedy identifikátor objektu bldg:BuildingInstallation vytvořený spodní větví zpracování polygonů vikýřů. Seskupení do kolekcí však musí být provedeno až po rozdělení na vodorovné a svislé polygony, protože každé kolekci polygonů pak bude přiřazen typ povrchu (stěna nebo střecha) a není možné, aby jedna kolekce obsahovala polygony různých druhů povrchu.

Zpracování svislých obvodových stěn je přímočaré. AttributeRenamer přejmenuje atribut obsahující identifikátory budov na gml_parent_id a Aggregator je podle tohoto atributu seskupí. Kolekce polygonů stěn tak budou náležet správným budovám. Větev vedoucí z obr. 5.5 směrem dolů posílá polygony stěn do části schématu sloužící k uzavírání objektů odspodu, o čemž pojednává kapitola 5.3.3. Pro tyto účely uloží AttributRenamer identifikátor budovy i do atributu ID objektu.

Portem <*Unfiltered*> centrální komponenty AttributeFilter prochází prvky, které nepatří do žádné z výše zmíněných tříd. Jsou to komíny, výtahy, větrání a klimatizace. Ty budou tvořit objekty bldg:BuildingInstallation. Na rozdíl od vikýřů však bude geometrie uložena přímo u těchto objektů, postačí tedy tato jedna větev. AttributeRenamer nastaví identifikátor budovy jako gmp_parent_id a identifikátor shluku jako gml_id (a také ID objektu pro stejné účely jako u stěn). Čára vedoucí od něj směrem dolů ven z obrázku vede stejně jako v případě stěn do části schématu sloužící k uzavírání objektů odspodu. V této části vytvořené polygony se pak vrací větví, která se

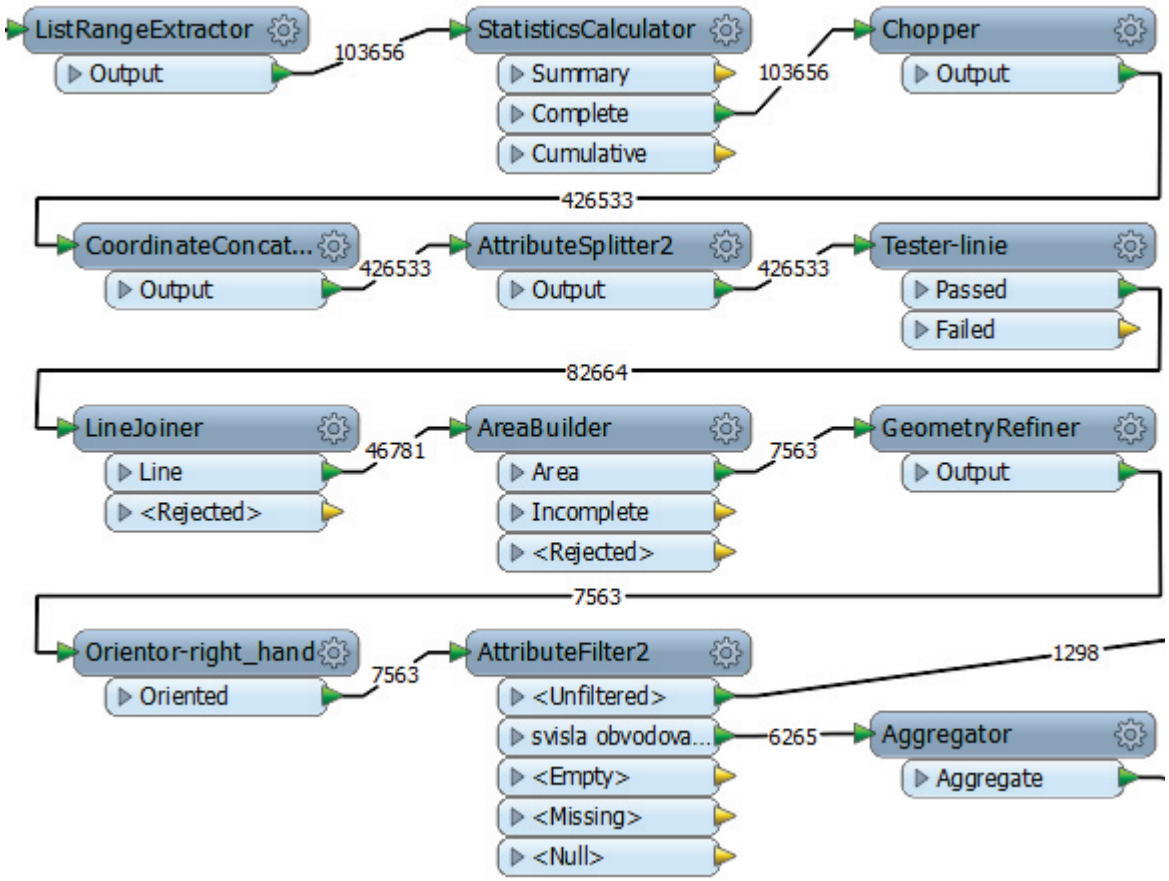
napojuje na spojnici s komponentou `Aggregator`, který spojuje polygony se stejnou hodnotou ID objektu do kolekcí.

### 5.3.3 Uzavírání objektů odspodu

Všechny 3D objekty v CityGML by měly být uzavřené, v digitálním modelu zástavby a zeleně hlavního města Prahy však nejsou. Budovy mají stěny a střechy, odspodu jsou ale otevřené. Zdrojová data tak neobsahují polygony, které by mohly být použity jako `bldg:GroundSurface`. Takové polygony je však možné z dostupných dat vytvořit, a to následujícím postupem:

1. Pro každý objekt (např. budovu) najít nejmenší hodnotu Z
2. Rozdělit polygony na jednotlivé úsečky
3. Vyfiltrovat úsečky, jejichž hodnota je rovna nejmenší hodnotě Z
4. Spojit vybrané úsečky a vytvořit z nich plochu
5. Vytvořenou plochu správně orientovat (opačně než střechy)

Celý tento proces je možné realizovat ve FME Workbench. Použitá série transformačních komponent je na obr. 5.6. Do počáteční komponenty této série vstupují polygony, které opouštějí část schématu na obr. 5.5 spojnicemi směřujícími dolů.




**obr. 5.6 Schéma uzavírání objektů odspodu**

Počáteční `ListRangeExtractor` uloží pro každý polygon hodnotu souřadnice Z jeho nejnižšího vrcholu do nového atributu. Získá ji ze seznamu výšek vytvořeného v části sloužící k nastavování orientace polygonů popsané v kapitole 5.3.1. Tyto minimální výšky použije

`StatisticsCalculator` k výpočtu nejmenší hodnoty Z ze všech polygonů se stejnou hodnotou ID objektu.

O rozdělení polygonů na úsečky se postará `Chopper` s definovaným maximálním počtem vrcholů prvku rovným 2.

Pro vyfiltrování při zemi ležících úseček jsou potřeba 3 komponenty. `CoordinateConcatenator` uloží výšky obou vrcholů úsečky do nového atributu a `AttributeSplitter` z tohoto atributu vytvoří dvouprvkový seznam. Následný Tester pak obě výšky porovná s hodnotou nejmenší výšky objektu zjištěnou komponentou `StatisticsCalculator`.

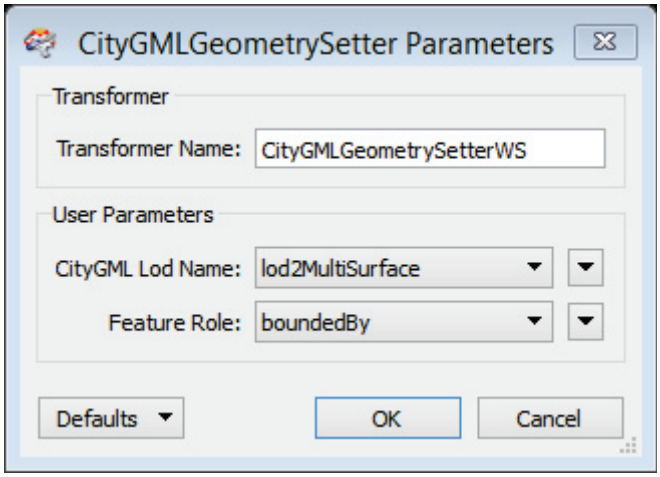
O pospojování úseček se postará `LineJoiner` a plochy z nich vytvoří `AreaBuilder`. Těmto komponentám je třeba nastavit jako atributy, podle kterých budou jednotlivé prvky seskupovat, kromě ID objektu i všechny další atributy, které budou potřeba při dalším zpracování. Konkrétně atribut určující typ prvky kvůli následnému filtrování a také identifikátor budovy pro správné zařazení do hierarchické struktury výsledného GML souboru. Protože `AreaBuilder` produkuje plochy ohraničené křivkami, je třeba zařadit ještě `GeometryRefiner`, který všechny ohraničující křivky polygonu nahradí jediným objektem `gml:LinearRing`.

Po nastavení orientace komponentou `Orientor` zbývá již jen roztřídění vzniklých polygonů podle toho, k jakému typu objektu patří. `AttributeFilter` pošle všechny polygony vzniklé z úseček svislých obvodových stěn spodním portem do komponenty `Aggregator`, kde jsou seskupené do kolekcí, které budou odspodu uzavírat budovy. Horním portem pošle všechny ostatní polygony. I ty budou seskupeny do kolekcí, ovšem komponentou `Aggregator` zobrazenou v pravém spodním rohu obr. 5.5. Ty budou zespodu uzavírat objekty typu `bldg:BuildingInstallation`.

### 5.3.4 Nastavení CityGML tříd a vazeb

Posledním úkolem transformačního nástroje je určit, které prvky budou patřit do které CityGML třídy, a nadefinovat vazby mezi nimi. Na obr. 5.8 je vidět, že bylo použito celkem 5 výstupních typů, které odpovídají CityGML třídám. Dva pro *`_CityObject`* (`Building` a `BuildingInstallation`) a tři pro `bldg:_BoundarySurface` (`bldg:WallSurface`, `bldg:RoofSurface` a `bldg:GroundSurface`).

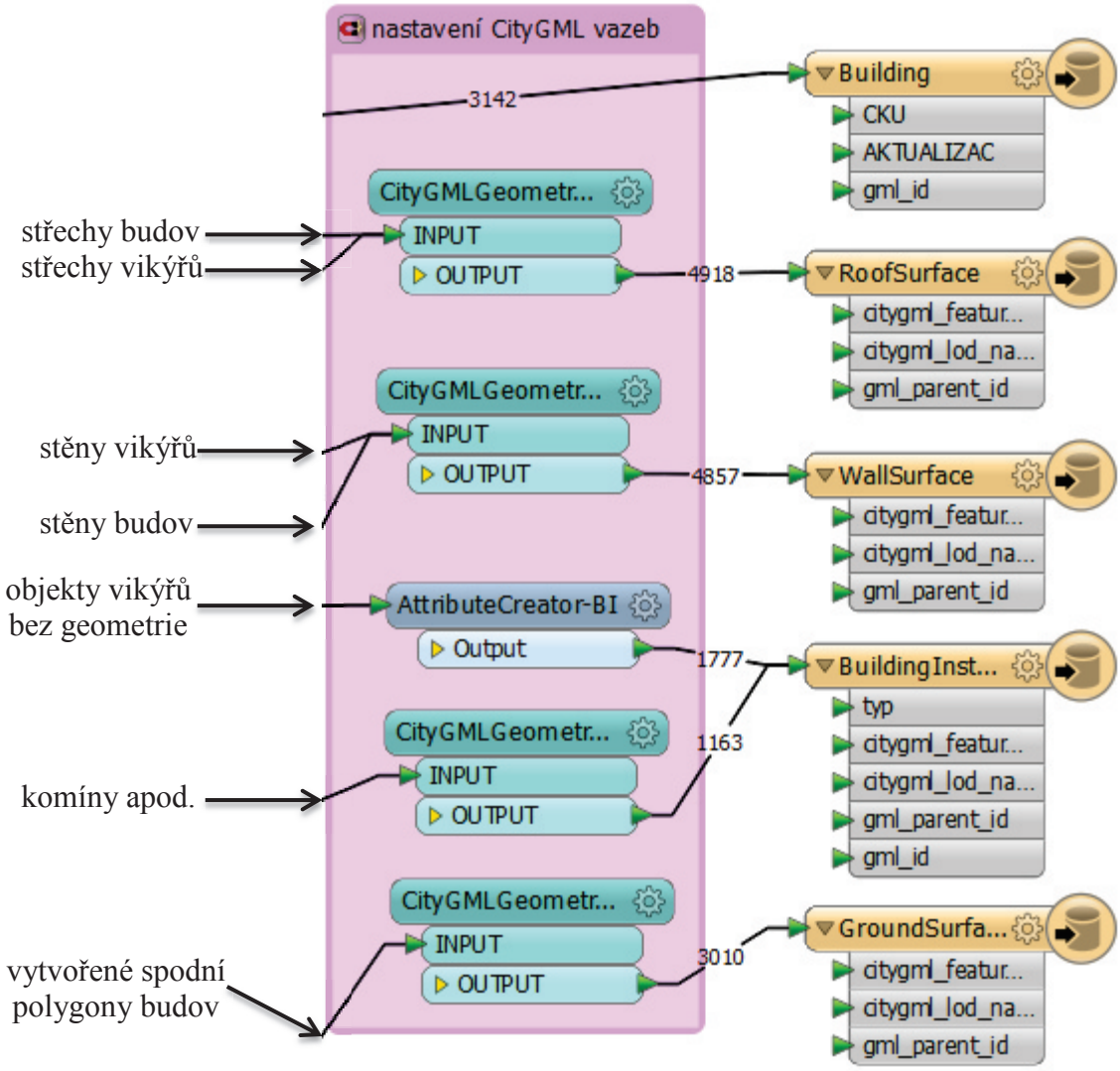
Do správných výstupních typů budou polygony směrovány na základě třídění provedeném komponentou `AttributeFilter` na obr. 5.5. Kromě toho je však třeba zajistit ještě použití správných vazeb mezi výstupním typům odpovídajícími třídami CityGML. K tomu slouží 2 parametry, které nastavuje `CityGMLGeometrySetter`. Parametr *CityGML lod Name* definuje vazbu od daného objektu na objekt geometrie. Určuje tak zároveň třídu použitou k uložení geometrie (tou může být *`gml:_Solid`*, `gml:MultiSurface`, *`gml:_Geometry`*). Parametrem *Feature Role* je pak definována vazba mezi daným objektem a jeho rodičem.




**obr. 5.7 CityGMLGeometrySetter pro stěny**

Pro objekty tříd odvozených z třídy `bldg:_BoundarySurface` má parametr *CityGML lod Name* vždy hodnotu `bldg:lod2MultiSurface` a parametr *Feature Role* pak nabývá hodnoty `bldg:boundedBy`, jak je možné snadno odvodit z obr. 3.4. Nastavení komponenty pro tyto objekty je na obr. 5.7.

Objektům třídy `bldg:BuildingInstallation`, které obsahují geometrii přímo, nastaví `CityGMLGeometrySetter` parametr *CityGML lod Name* na hodnotu lod2Geometry a parametr *Feature Role* na hodnotu `bldg:outerBuildingInstallation`. Těm objektům této třídy, které vlastní geometrii nemají (ta je obsažena až v objektech tříd odvozených z `bldg:_BoundarySurface`), postačí nastavit parametr *Feature Role*. Místo komponenty `CityGMLGeometrySetter` je tak použit `AttributeCreator`.



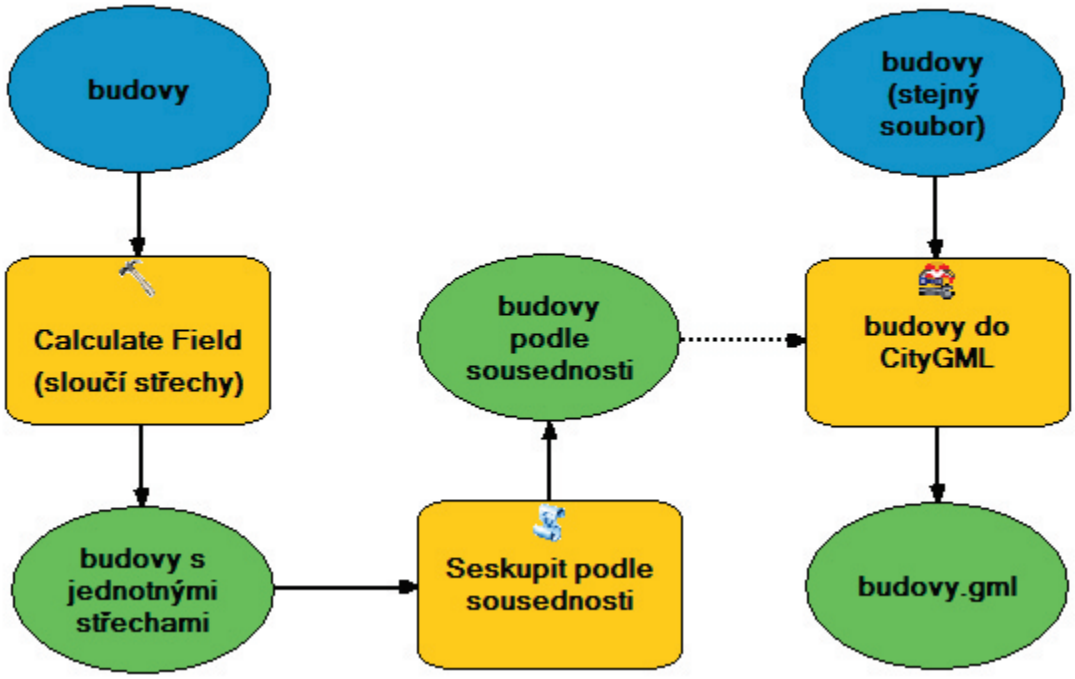

**obr. 5.8 Schéma nastavování CityGML tříd a vazeb**

Pro objekty třídy `bldg:Building` není žádná z těchto komponent potřeba. Geometrii (a tudíž ani vazbu na ní) nemají a vazbu `cityObjectMember` z kořenového elementu `CityModel` doplní program budovám sám.

Finální část schématu transformace sloužící k nastavování těchto vazeb je znázorněna na obr. 5.8, který navazuje na obr. 5.5. Pro snadnější orientaci jsou vstupní spojnice popsány. Pro ještě lepší orientaci v celém schématu lze nahlédnout do přílohy 3. Do horního výstupního typu `Building` vstupují geometrie zbavené základní objekty budov.

### 5.3.5 Kompletní proces

Jak již bylo zmíněno, proces převodu Digitálního modelu zástavby a zeleně hlavního města Prahy do CityGML v aplikaci FME Workbench vyžaduje, aby vstupní data obsahovala nejen identifikátory budov (které jsou v Digitálním modelu zástavby a zeleně přítomny), ale také identifikátory jednotlivých komínů, vikýřů a dalších drobných objektů na budovách. O doplňování těchto identifikátorů do dat pojednává kapitola 4. V její části 4.4 je pak popsán problém s různými typy střech a jeho řešení. Celý proces převodu budov do CityGML je pak zahrnut v ArcGIS modelu, který zobrazuje **Chyba! Nenalezen zdroj odkazů.**.




**Obrázek 5.9 Celkový proces převodu budov do CityGML**

Nejprve je nástrojem *Calculate Field*, jehož nastavení je na obr. 4.7, sjednocena klasifikace střech. Poté je spuštěn nástroj *Seskupit podle Sousednosti* popsaný v podkapitole 4.3, který k jednotlivým polygonům doplní identifikátory shluků, do kterých náleží. Teprve nakonec přijde řada na v aplikaci FME Workbench vytvořený nástroj popisovaný v této kapitole, který data převede do formátu CityGML. Jeho výstupem je jediný GML soubor.

Protože tento typ nástroje umožňuje jako vstupní parametr pouze obecný soubor a nikoliv třídu prvků, jak vyžadují ostatní v modelu použité nástroje, obsahuje model druhý parametr, který však musí být vždy nastaven na stejný soubor (přesněji na soubor, pro který již byla spuštěna předchozí část modelu). Spojení přerušovanou čarou značí, že nástroj *budovy do CityGML* bude spuštěn až poté, co úspěšně proběhnou předchozí 2 nástroje. Soubor, který je jeho vstupem, tak bude připraven na transformaci do CityGML.

Pro převod vybraného výřezu vrstvy budov Digitálního modelu zástavby a zeleně hlavního města Prahy tedy stačí spustit připravený model, přičemž oba jeho parametry musí být nastaveny na stejný soubor. Model je spolu s vytvořeným nástrojem *Seskupit podle sousednosti* součástí toolboxu, který se nachází na přiloženém CD. Aktuální verze ArcGIS je však distribuována se starší verzí aplikace FME Workbench, ve které bohužel není možné otevřít soubor upravený v nejnovější verzi. Proto je vytvořený nástroj pro finální převod do CityGML uložen na CD rovněž samostatně, aby jej bylo možné otevřít přímo v aplikaci FME Workbench.

# 6 Závěr

V této práci byl řešen problém tvorby dat ve formátu CityGML z existujících plochých dat GIS. Cílem bylo identifikovat v datovém modelu CityGML třídy vhodné pro reprezentaci Digitálního modelu zástavby a zeleně hlavního města Prahy a provést převod těchto GIS dat do CityGML.

Výsledkem práce je navržený proces převodu dat do CityGML a také nástroje, kterými je možné tento proces realizovat. Konkrétně jde o nástroj pro identifikaci shluků polygonů podle jejich vzájemné sousednosti implementovaný v jazyce Python a převodní nástroje pro budovy a terén realizované v aplikaci FME Workbench. Za drobný přínos práce lze snad považovat i prozkoumání a popis možností využití FME pro převody mezi různými formáty geografických dat. Možnosti tohoto programu však dalece přesahují její rozsah.

Některé problémy nutně spojené s převodem mezi natolik rozdílnými formáty se podařilo vyřešit a výstupem jsou tak korektní GML soubory. Pro zcela správná CityGML data by však bylo potřeba ještě mnoho další práce. Problematika orientace stěn si zaslouží lepší řešení a zdokonalit by bylo vhodné i uzavírání objektů.

Největším problémem je však topologické čištění dat, ve kterých se vykytuje mnoho chyb. Stěny budov končívají v různé výšce, což znemožňuje automatizované uzavření těchto staveb, některé budovy na sebe nenavazují a vyskytují se i případy chybné klasifikace. Jednotlivé objekty by na sebe měly navazovat a nikoliv se překrývat. Vyřešení všech těchto problémů a zahrnutí CityGML topologie do výstupů by bylo jistě žádoucí. Takové činnosti by však svým rozsahem zřejmě přesáhly souhrn ostatních úkonů potřebných pro převod do CityGML.

# Zdroje


[1] *OGC City Geography Markup Language (CityGML) Encoding Standard*. Verze 2.0.0. Open Geospatial Consortium, 4. 4. 2012. Dostupné z http://www.opengis.net/spec/citygml/2.0.

[2] 3D model - Digitální model zástavby a zeleně. 14. 12. 2010 [citováno 27. 2. 2014]. Dostupné z http://www.geoportalpraha.cz/cs/clanek/6/3d-model-digitalni-model-zastavby-a-zelene

[3] *Digitální mapa veřejné správy hl. m. Prahy*. Praha: Institut plánováni a rozvoje hlavního města Prahy, 2013 [citováno 27. 2. 2014]. Dostupné z http://www.geoportalpraha.cz/uploads/assets/DMVSP_Priloha1_usneseni.pdf

[4] *PyDev manual* [online]. Poslední revize 20. 5. 2014 [citováno 26. 5. 2014]. Dostupné z http://pydev.org/manual.html.

[5] *ArcGIS Help* [online]. Verze 10.2. ESRI, 16. 12. 2013. Dostupné z http://resources.arcgis.com/en/help/main/10.2/.

[6] *FME Workbench* [online]. Verze 2014. Safe Software, 2014. Dostupné z http://docs.safe.com/fme/html/FME_Workbench/Default.htm.

[7] *FME Workbench Transformers* [online]. Verze 2014. Safe Software, 2014. Dostupné z http://docs.safe.com/fme/html/FME_Transformers/Default.htm.

[8] *Schema Central* [online]. Dostupné z http://www.schemacentral.com/index.html.

[9] *FME Readers and Writers* [online]. Verze 2014. Safe Software, 2014. Dostupné z http://docs.safe.com/fme/html/FME_ReadersWriters/Default.htm.

# Přílohy

## Příloha 1 – zdrojový kód skriptu pro identifikaci objektů

```
# -*- coding: utf-8 -*-
'''
Created on 6. 4. 2014
Seskupí polygony vstupní třídy prvků do shluků na základě jejich sousednosti.
@author: Jiří Fiedler
'''
import arcpy
import locale
from operator import itemgetter
from os import path
import time

tridaPrvku = arcpy.GetParameterAsText(0)
sloupecTypu = arcpy.GetParameterAsText(1)
sloupecShluku = arcpy.ValidateFieldName(arcpy.GetParameterAsText(2),
path.dirname(tridaPrvku))

sousedi = {} # prázdný slovník sousedů: {'prvek': ['soused1', 'soused2',...]}
shluky = {} #  prázdný slovník shluků: {'prvek': 'shluk'}
ekvivalentniShluky = set() # prázdná množina seznamů ID označujících stejný shluk
hodnotaUkazatelePrubehu = 0 # udržuje aktuální pozici ukazatele průběhu

def vypisZpravu(zprava):
    """Vypíše na obrazovku aktuální čas a text v systémovém kódování, aby se
zobrazil správně na počítači s lokalizovanou verzí Windows."""
    cas = time.strftime("%H:%M:%S ")
    arcpy.AddMessage(cas + zprava.encode(locale.getpreferredencoding()))

def zobrazPopisUkazatele(popis):
"""Ukáže popis na obrazovce ukazatele průběhu s použitím systémového kódování."""
    arcpy.SetProgressorLabel(popis.encode(locale.getpreferredencoding()))

def predchoziSousedi(vybranaTrida, sousedi):
"""Přidá do slovníku seznamy předchozích sousedů všech prvků vstupní třídy prvků.
Slovník obsahuje pro každý klíč neprázdný seznam sousedů s ID nižším než ID klíče.
Ne každý prvek se v tomto slovníku vyskytuje jako klíč (sousednost 3-2 je
zahrnuta, sousednost 2-3 nikoliv)."""

    vypisZpravu(u"tvorba slovníku předchozích sousedů")

     cestaKTabulce = "in_memory" + path.sep + "tabulka"
     # s in_memory workspace je výpočet rychlejší (neukládá se na disk)

    tabulka = arcpy.analysis.PolygonNeighbors( vybranaTrida,
                                               cestaKTabulce,
                                               sloupecShluku,
                                               "NO_AREA_OVERLAP",
                                               "NO_BOTH_SIDES").getOutput(0)
```

```
    with arcpy.da.SearchCursor(tabulka,
                               ["src_" + sloupecShluku, "nbr_" + sloupecShluku])
                               as tabulkaSousednosti:
        for radka in tabulkaSousednosti:
            if (radka[1] not in sousedi):
                sousedi[radka[1]] = list() # přidá klíč s hodnotou prázdného seznamu
            sousedi[radka[1]].append(radka[0]) # přidá src_ID do sousedů nbr_ID
    arcpy.management.Delete(tabulka) # dále není potřeba

def posunUkazatel(krok):
    """Posune ukazatel průběhu o daný krok."""
    global hodnotaUkazatelePrubehu
    hodnotaUkazatelePrubehu += krok
    arcpy.SetProgressorPosition(int(hodnotaUkazatelePrubehu))

# inicializace
sloupecID = arcpy.Describe(tridaPrvku).OIDFieldName
pocetPrvku = int(arcpy.management.GetCount(tridaPrvku).getOutput(0))
arcpy.SetProgressor("step", "", 0, 1000, 1) # úvodní nastavení ukazatele průběhu

# přidání nového sloupce pro čísla shluků a jeho prozatimní naplnění ID
vypisZpravu(u"přidávání nového sloupce")
arcpy.management.AddField(tridaPrvku, sloupecShluku, "LONG")
posunUkazatel(10)
zobrazPopisUkazatele(u"přidávání nového sloupce")
arcpy.management.CalculateField(tridaPrvku, sloupecShluku, "!" + sloupecID + "!")
posunUkazatel(40) # posune ukazatel o 40 bodů na hodnotu 50

if (len(sloupecTypu) > 0): # je-li nějaký sloupec typů definován

    # zjištění možných hodnot ve sloupci typů a volba podmínky podle datového typu
    with arcpy.da.SearchCursor(tridaPrvku, [sloupecTypu]) as trida:
        typyPrvku = set(radek[0] for radek in trida)
    if arcpy.ListFields(tridaPrvku, sloupecTypu)[0].type == "String":
        podminkaSQL = u"{0} = '{1}'"
    else:
        podminkaSQL = u"{0} = {1}"

    # vytvoření slovníku předchozích sousedů
    krok = 350.0/len(typyPrvku)/3 # pro každý typ se ukazatel průběhu změní 2×
    for typPrvku in typyPrvku:
        vypisZpravu(u"typ prvku: {}".format(typPrvku))
        zobrazPopisUkazatele(u"probíhá výběr...")
        cestaKeTride = "in_memory" + path.sep + "trida"
        vybranaTrida = arcpy.analysis.Select(tridaPrvku, cestaKeTride,
podminkaSQL.format(sloupecTypu, typPrvku)).getOutput(0)
        posunUkazatel(krok)
        predchoziSousedi(vybranaTrida, sousedi)
        arcpy.management.Delete(vybranaTrida)
        posunUkazatel(2*krok) # výpočet sousedů trvá déle než výběr
else: # volitelný parametr sloupec typů nepoužit => následuje tvorba slovníku
    predchoziSousedi(tridaPrvku, sousedi)
    posunUkazatel(350.0)
```

```
# přiřazování ID shluků k jednotlivým prvkům
vypisZpravu(u"přiřazování ID shluků k jednotlivým prvkům")
krok = 100.0/pocetPrvku # celkem se v této fázi posune ukazatel o 100 bodů
popisDialogovehoOkna = u"přidávání shluku {0}"
with arcpy.da.SearchCursor(tridaPrvku, [sloupecID]) as trida:
   for prvek in trida:
      if (prvek[0] in sousedi): # prvek sousedí s aspoň jedním předchozím prvkem
         sousediPrvku = sousedi[prvek[0]]
         shluky[prvek[0]] = shluky[sousediPrvku[0]]
         if (len(sousediPrvku) > 1): # prvek má víc sousedů
            mnozinaEkvivalence = set()
            for soused in sousediPrvku:
               mnozinaEkvivalence.add(shluky[soused])
            if (len(mnozinaEkvivalence) > 1): # sousedi patří do různých shluků
               ekvivalentniShluky.add(tuple(sorted(mnozinaEkvivalence)))
      else:
         if (len(shluky) == 0):
            shluky[prvek[0]] = 1 # v případě prázdného slovníku shluků
         else:
            shluky[prvek[0]] = max(shluky.values()) + 1 # přiřadí nový shluk
         zobrazPopisUkazatele(popisDialogovehoOkna.format(len(shluky)))
      posunUkazatel(krok)

# přečíslování shluků podle tabulky ekvivalence barev
vypisZpravu(u"přečíslovávání shluků podle tabulky ekvivalence")
seznamEkvivalentnichShluku = sorted(ekvivalentniShluky,
                                    key = itemgetter(0),
                                    reverse = True)
pocetEkvivalentichShluku = len(seznamEkvivalentnichShluku)
krok = 495.0/pocetEkvivalentichShluku # v této fázi se posune ukazatel o 495 bodů

popisDialogovehoOkna = u"přečíslování podle množiny ekvivalence {0} z {1}"

for i, enticeEkvivalence in enumerate(seznamEkvivalentnichShluku): # iterace přes
všechny entice, i je jen index pro popis
   prvniShluk = 0 # první shluk z entice, jehož hodnotou budou nahrazeny ostatní
   for shlukEkvivalence in enticeEkvivalence:
      if prvniShluk == 0: # první shluk z entice je třeba vždy vynechat z iterace
         prvniShluk = shlukEkvivalence
      else: # pro všechny shluky z entice kromě prvního
         for (prvek, shlukPrvku) in shluky.iteritems():
            if (shlukPrvku == shlukEkvivalence):
               shluky[prvek] = prvniShluk
   posunUkazatel(krok)
   zobrazPopisUkazatele(popisDialogovehoOkna.format(i, pocetEkvivalentichShluku))

# zápis čísel shluků do příslušného sloupce třídy prvků (přepíše hodnoty ID, které
už nyní nejsou třeba - rovnají se hodnotám ve sloupci sloupecID)
vypisZpravu(u"zápis finálních hodnot")
with arcpy.da.UpdateCursor(tridaPrvku, [sloupecID, sloupecShluku]) as trida:
   for prvek in trida:
      prvek[1] = shluky[prvek[0]]
      trida.updateRow(prvek)
arcpy.SetProgressorPosition(1000)
```

# Příloha 2 – zdrojový kód validační třídy

```
import arcpy
class ToolValidator(object):
   """Třída pro validaci hodnot parametrů a správu schématu výstupu nástroje
    seskupitPodleSousednosti."""

   def __init__(self):
      """Inicializace seznamu parametrů."""
      self.params = arcpy.GetParameterInfo()

   def initializeParameters(self):
      """Inicializace parametrů po otevření dialogového okna nástroje."""
      self.params[3].parameterDependencies = [0, 2]
      self.params[3].schema.clone = True
      return

   def updateParameters(self):
      """Aktualizace schématu výstupní třídy prvků po každé změně hodnoty
parametru název sloupce (umožňuje použití nástroje v modelech)."""
      if self.params[0].value and self.params[2].value:
         poleShluk = arcpy.Field()
         poleShluk.name = self.params[2].value
         self.params[3].schema.additionalFields = [poleShluk]
      return

   def updateMessages(self):
      """Dodatečná validace názvu sloupce po interní validaci."""
      if not self.params[2].value.isalnum():
         self.params[2].setErrorMessage("Pro název sloupce jsou přípustné pouze
alfanumerické znaky.")
      elif len(self.params[2].value) > 10:
         self.params[2].setErrorMessage("Název nesmí být delší než 10 znaků.")
      elif self.params[0].value:
         if self.params[2].value.encode("utf-8") in [sloupec.name
                     for sloupec in arcpy.Describe(self.params[0].value).fields]:
            self.params[2].setErrorMessage("Sloupec s tímto názvem již existuje.")
      else:
         self.params[2].clearMessage()
      return
```

# Příloha 3 – schéma převodu budov do CityGML

# Obsah přiloženého CD

- `Modelování budov a terénu 3D modelu Prahy podle standardu CityGML.pdf` – text diplomové práce totožný s tištěnou verzí
- `seskupitPodleSousednosti.py` – skript pro identifikaci shluků vzájemně sousedících polygonů
- `seskupitPodleSousednostiValidator.py` – validační třída pro ArcGIS nástroj obsahující
- `DMR do CityGML.fmw` – workspace aplikace FME Workbench pro převod DMR do CityGML, vytvořený nástroj lze ve FME dále upravovat
- `budovy do CityGML.fmw` – workspace aplikace FME Workbench pro převod budov do CityGML, vytvořený nástroj lze ve FME dále upravovat
- `diplomová práce.tbx` – ArcGIS toolbox obsahující oba výše uvedené nástroje, nástroj *Seskupit podle sousednosti* využívající stejnojmenný skript a model pro zpracování budov, který ostatní nástroje využívá
- `schéma transformace budov.PNG` – schéma nástroje pro převod budov v plném rozlišení
- `SHP terénu` – složka obsahující shapefile budov (zdroj převodu do CityGML)
- `SHP terénu` – složka obsahující shapefile terénu (zdroj převodu do CityGML)
- `DMR.gml` – výsledná CityGML podoba DMR z Digitálního modelu zástavby a zeleně hlavního města Prahy
- `budovy.gml` – výsledná CityGML podoba budov z Digitálního modelu zástavby a zeleně hlavního města Prahy
- `obsah CD.txt` – tento obsah