Rapport - AP2

SmartFlow

Dokumenthistorik:

Version	Initialer	Datum	Kommentar
1.0	LWi, KWe, LHe, ANi, HBl	2021-05-21	Rapport för granskning av projektets parter

Innehållsförteckning

Sammanfattning

Detta dokument beskriver det arbete som genomförts inom ramen för arbetspaket 2 (AP2) i projektet SmartFlow samt de slutsatser som dragits under arbetets gång. Dokumentet ger även rekommendationer för vidare arbete i nästa arbetspaket (AP3).

Syftet med arbetspaketet har varit att i möjligaste mån verifiera ansatsen med SmartFlow-projektet.

Baserat på vad vi gjort i AP2 och de slutsatser vi har dragit finns det inget som motsäger den grundläggande ansatsen med SmartFlow-projektet. Vi rekommenderar att vi arbetar vidare med projektet i AP3.

I AP2 har vi:

- Köpt in en licens av GraphDB och etablerat en SmartFlow Server utvecklingsmiljö i Azure
- Genomfört tre testfall med konverteringar och transformeringar, till och från RDF, av data i olika format
    - Testfall med data från Statens Vegvesen, Trafikverket och Ramboll. När det gäller indata i form av IFC-filer har Trimble genererat dessa.
- Skapat demo-struktur för metadata
- Skapat en enkel webb-applikation för att kunna demonstrera metadata-strukturen
- Skaffat api-nycklar för CoClass och börjat skissa på en lösning med en ”kanonisk modell”
- Startat ett arbete med att detaljera kravbilden när det gäller processer och flöden
- Startat ett arbete med att definiera kvalitetskrav för lösningen
- Ökat arbetet med GraphDB för att öka kompetensen och förståelsen för produktens möjligheter och begränsningar

Utifrån lärdomarna från arbetet i AP2 föreslår vi att vi i AP3 fortsätter arbeta med att:

- Definiera scope för demonstrationer
- Ta fram representativa testfall
- Förfina, detaljera och prioritera use cases/user stories
- Utveckla och beskriva en arkitektur för lösningen
    - Definiera egenskapskrav för arkitekturen
    - Undersöka effekterna och behoven av olika exekveringsmiljöer
    - Ta fram arkitekturella modeller
    - Undersöka och testa användandet av kart- och 3D-tjänster
- Utveckla lösning för demonstration; ett demo-system

- Som en del av lösningen undersöka och testa några explicita utmaningar och hypoteser, såsom
    - Geometriförenkling
    - Mappning mot topologiskt nät
    - Hantering av koordinatsystem/koordinattransformationer
    - Principer för datatransformationer
    - Principer för lagring och strukturering av data och metadata
    - Principer för verifiering och validering av data

1 Inledning

1.1 Syfte och målgrupp

Detta dokument beskriver det arbete som genomförts inom ramen för arbetspaket 2 i projektet SmartFlow samt de slutsatser som dragits under arbetets gång. Dokumentet ger även rekommendationer för vidare arbete i nästa arbetspaket (AP3).

Syftet med arbetspaketet har varit att i möjligaste mån verifiera ansatsen med SmartFlow-projektet:

- Identifiera behov och krav för denna typ av lösning för hantering av dataflöden mellan projekt (investering och inmätning) och system för anläggningsförvaltning
- Identifiera icke-funktionella krav för denna typ av lösning
- Verifiera att teknologin (Länkade data/Semantisk web) är ett möjligt sätt att hantera dessa dataflöden, på ett sätt som är frikopplat från applikationsspecifika (”stuprörs-”) lösningar
- Få en uppfattning om tekniska begränsningar med teknologin

Dokumentet är primärt ägnat åt projektets parter samt styrgrupp som ett underlag för beslut kring fortsatt arbete i projektet.

1.2 Referenser

1. Rapport arbetspaket 1, SmartFlow
2. https://semiceu.github.io/GeoDCAT-AP/drafts/latest/
3. https://www.w3.org/2011/rdf-wg/wiki/Main_Page
4. https://www.w3.org/TR/rdf-schema/
5. https://www.w3.org/TR/owl-guide/
6. https://www.w3.org/TR/shacl/
7. https://www.w3.org/TR/sparql11-query/
8. https://www.ogc.org/standards/geosparql
9. https://technical.buildingsmart.org/standards/ifc/
10. https://technical.buildingsmart.org/standards/ifc/ifc-formats/ifcowl/
11. https://coclass.byggtjanst.se/login
12. https://www.kartverket.no/geodataarbeid/standardisering/sosi-standarder2
13. https://www.opentnf.org/
14. https://github.com/vegvesen/NVDB-Datakatalogen/tree/master/OWL
15. https://www.ontotext.com/products/graphdb/
16. https://www.esri.com/content/dam/esrisites/sitecore-archive/Files/Pdfs/library/whitepapers/pdfs/shapefile.pdf

1.3 Definitioner och förkortningar

Term/förkortning	*Definition*
RDF	*Resource definition framework*
RDFS	*Resource definition framework schema*
OWL	*Web ontology language*
SHACL	*Shape's constraint language*
IFC	*Industry foundation classes*
NVDB	*Nationell vägdatabas (Trafikverket)/Nasjonal vegdatabank (Statens Vegvesen)*
GUS	*Gemensamt underhållsstöd (Trafikverket)*
BaTMan	*Bridge and tunnel management (Trafikverket)*
Ontologi	*En formell beskrivning av koncept och relationer inom en domän. Synonymt med TBOX.*
TBOX	*Den del av en kunskapsgraf som beskriver terminologi*
ABOX	*Den del av en kunskapsgraf som beskriver fakta i termer av innehållet i TBOX*
Kunskapsgraf	*En kunskapsbas som använder en grafstrukturerad datamodell eller topologi för att integrera data. Kunskapsdiagram används ofta för att lagra sammanlänkade beskrivningar av enheter - objekt, händelser, situationer eller abstrakta begrepp - med friformsemantik. [Wikipedia]*
Länkade data	*En metod för att publicera strukturerad data på ett sätt som gör det möjligt att följa kopplingar mellan informationsobjekt över flera källor. Länkade data bygger på webbstandarder som HTTP och URI:er men istället för att använda dem för att presentera information för människor är syftet att göra data läsbart för maskiner. Målet är att möjliggöra för vitt skilda aktörer att beskriva sin information på ett enhetligt sätt och därmed förenkla återanvändning och interoperabilitet*
Semantisk web	*Beskriver metoder och teknik för att möjliggöra för maskiner att förstå innebörden eller "semantiken" i informationen på webben. Tekniken bygger på länkade data.*
Reasoning	*En "Reasoner" är mjukvara som kan härleda logiska konsekvenser från en mängd fakta eller axiom.* *Anm: För SmartFlow kan denna typ av mjukvara exempelvis användas för att automatiskt klassificera om objekt och egenskaper utifrån att ontologier länkats samman via relationer med en definierad och standardiserad logik.*

2 Genomförande av arbetspaket 2

2.1 Förutsättningar

Såsom beskrivs i kapitel 1.1 så har syftet med arbetspaket 2 varit att i största möjliga omfattning verifiera ansatsen med SmartFlow.

För arbetspaket 2 har vi valt att fokusera arbetet på några grundantaganden:

- Vi ska testa hantering av både IFC- och shape-data inom ramen för AP2
- Vi ser metadata som en essentiell komponent i denna typ av lösning där användare snabbt och enkelt kan få en översikt över vilka data som finns tillgängliga och vad de har för status.
- Vi är öppna för möjligheten att man rent tekniskt behöver kunna samarbeta väl med andra produkter specialiserade på exempelvis 3d-visualisering och karttjänster
- Vi har begränsade möjligheter i detta arbetspaket att explicit testa mot specifika konsumenter, t ex NVDB i Sverige/Norge, GUS, BatMan etc så utdata riktad specifikt mot dessa behöver simuleras
    - För NVDB-Norge har en ontologi (nvdb-owl - https://github.com/vegvesen/NVDB-Datakatalogen/tree/master/OWL) erhållits som är en spegling av datakatalogen
    - För Sverige används TNE där vi kan definiera datakataloger som motsvarar tänkbara konsumenter (t ex NVDB eller GUS)
- Vi vill testa vilka möjligheter som kan erhållas med hjälp av OGC GeoSPARQL - https://www.ogc.org/standards/geosparql
- För metadata vill vi testa befintliga standarder som finns tillgängliga med semantisk webb-teknik. I första hand vill vi titta på DCAT-ontologin (https://www.w3.org/TR/vocab-dcat-3/)
- Vi är ovana med hantering av triple stores for hantering av kunskapsgrafer och behöver få en känsla för strukturering med avseende på prestanda, säkerhet och funktionalitet

2.2 Aktiviteter

2.2.1 Testmiljö

Microsoft Teams

För samarbete inom projektet med samtliga parter används Microsoft Teams (Team: SmartFlow). Dokumentdelning sker via fliken ”Gemensamma dokument” där följande folderstruktur är upplagd:

- Arbetsdokument
- Leverabler
- Möten
- Officiella dokument
- Presentationer

- Testdata
    - Ramboll
    - SVV_Hellefossen
    - TRV_KungensKurva

Microsoft DevOps

För Trionas arbete används Microsoft DevOps där vi primärt hanterar programkod och aktivitetsplanering.

Microsoft Azure

I Microsoft Azure har vi etablerat en utvecklings-/testmiljö i form av en Windows Server 2019 (4 kärnor/16GB och ca 150 GB disk).

I den miljön har vi installerat en instans av Ontotext GraphDB vilken vi använt för utveckling och test under AP2. Tanken är att vi där ska installera de komponenter (egenutvecklade och tredje-parts) som behövs för att kunna demonstrera SmartFlow som ett resultat av AP3.

Tack vare molnarkitekturen (Infrastructure as a Service, IaaS) är det lätt att skala upp/ut utvecklingsmiljön vid behov inom ramen för de ekonomiska förutsättningarna.

Ontotext GraphDB

Som motor och lager för RDF-data används GraphDB (https://www.ontotext.com/products/graphdb/) från Ontotext (https://www.ontotext.com/). Projektet har köpt en standard-licens av produkten som installerats i Microsoft Azure. Projektet har via avtal även tillgång till 3 dagars development support som ännu ej utnyttjats.

GraphDB är Java-baserat och kräver en installation av Java JRE/JDK.

Ett par motiv för val av GraphDB är:

- GraphDB har använts av delar av projekt-teamet tidigare vilket har gjort det lättare att komma igång
- GraphDB har stöd för Spatiala frågor genom GeoSparql vilket kan ha stor betydelse då data som ska hanteras av SmartFlow till största del är geografiska till sin natur och kan bl.a. underlätta implementation av stödjande användningsfall där man vill söka efter data genom en karta

IFCtoRDF

För att konvertera IFC-filer från STEP-format (enligt ISO 10303-21) till RDF (Turtle) enligt IfcOWL använder vi en open source-komponent IFCtoRDF (https://github.com/pipauwel/IFCtoRDF).

Komponenten körs som en fristående applikation från kommandoraden med kommando:
java -jar IFCtoRDF-0.4-SNAPSHOT-shaded.jar path/to/inputfile.ifc path/to/outputfile.ttl

TNE Generate Feature

Generate Feature är en tjänst som TNE erbjuder via REST-api eller ett webb-GUI som utifrån tabell-data innehållande geometri (geografiskt läge som punkter, linjer, ytor) samt egenskapsdata (t ex i shape-format) kan skapa nätanknutna företeelser enligt definitioner i en datakatalog.

Tjänsten innehåller grovt sett två funktioner:

- Mappning av egenskapskolumner till företeelseattribut
- Geometrisk matchning av geometrier mot nät för att generera nätanknytning

Den sistnämnda funktionen finns tillgänglig som en separat funktion som också är av intresse för SmartFlow i de fall att RDF-data har geometri och där nätanknytning behöver beräknas fram.

Funktionaliteten har använts i testfallet för vägräcken, vägmärken mm som beskrivs i kapitel 4.3.

Utvecklingsspråk och API:er

Som utvecklingsspråk har vi använt C#/.NET5, motiverat bl.a. av att projektet har störst kompetens runt C# samt möjligheten att använda vissa api:er. När det gäller .NET5 (-->.NETX) är det det ramverk som Microsoft ser som huvudspåret för .NET-utveckling. För front end-utveckling används vid behov också JavaScript.

För att hantera RDF-data programmatiskt använder vi open source-biblioteket dotNetRDF (https://dotnetrdf.org).

För att hantera OpenTNF-data programmatiskt använder vi open source-biblioteket OpenTNF-library (https://github.com/OpenTNF/OpenTNF-library).

Dessa val kan revideras under AP3 vid behov.

2.2.2 Testfall

De testfall som genomförts i arbetspaket 2 listas i nedanstående tabell. En mera detaljerad beskrivning finns i kapitel 4.

Testfall	Format	Innehåll	Resultat
Stikkrenner - **Hellefossen** **Statens Vegvesen**	IFC 2x3 IFC 4	Stikkrenner i form av IfcFlowSegment (IFC2x3) och IfcPipeSegment (IFC4). Egenskaper i PropertySet: - PredefinedType - InletType - MainUsage - MaterialType - OutletType - Cross-SectionShape - InnerDiameter - OperationYear Geometri som IfcFacetedBrep eller IfcPolyline	Inläst i GraphDB Konverterat till nvdb-owl och exporterat till excel. Metadata skapat

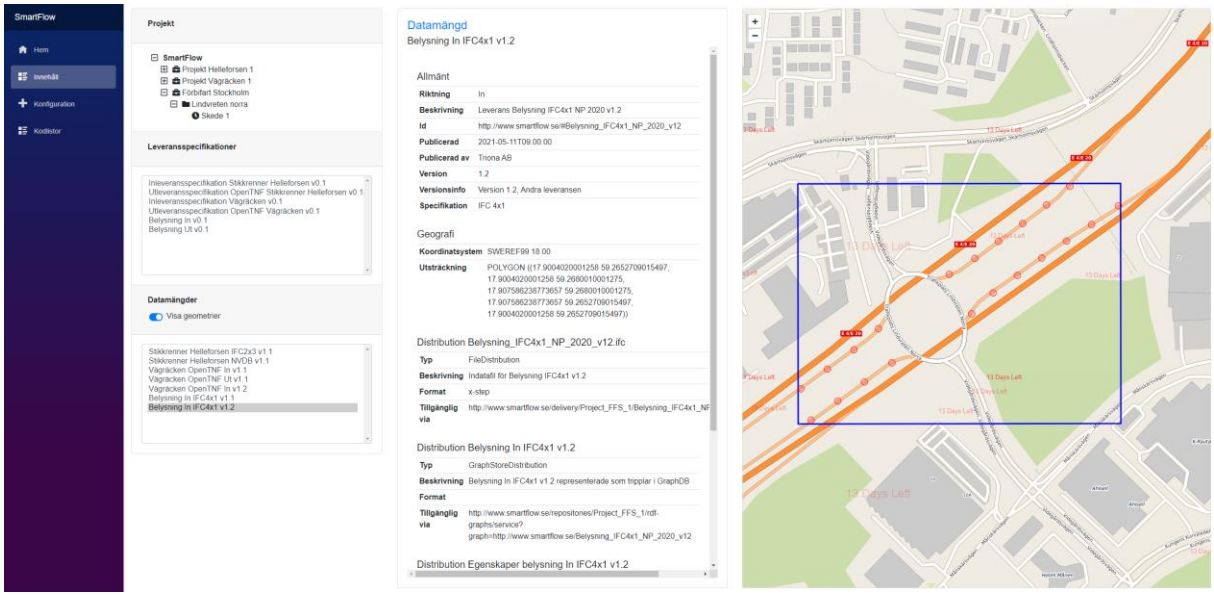
Belysningsstolpar – Lindvreten Norra **Trafikverket**	IFC 4	Belysningsstolpar i form av IfcMember (stolpe) och IfcLightFixture (Armatur) Egenskaper i PropertySet: - Komponent-id Egenskaper i separat excel enligt TRV laddmall. Geometri i version 1: IfcFacetedBrep Geometri i version 2: IfcCircle	Inläst i GraphDB Metadata skapat
Vägräcken **Vägmärken**	Shape		Inläst i GraphDB Metadata skapat

2.2.3 Metadata

Metadata i detta kapitel innebär data om data inklusive beskrivning av leveransspecifikationer samt den struktur data organiseras i.

Vi har:

- Skapat demo-struktur för metadata
- Utvecklat en enkel webb-applikation för att demonstrera metadata



Figur 1 - Webb-applikation för att demonstrera metadata

Bakgrund

Baserat på arbetet i AP1 finns ett förväntat behov att i SmartFlow kunna organisera data i projekt, delprojekt och skeden i en hierarki. Där framkommer också ett behov av att kunna registrera leveransspecifikationer för verifiering av förväntade framtida in- och utleveranser samt att kunna beskriva de datamängder som levererats in, transformerats och konsumerats. Det finns också ett förväntat behov av att kunna registrera aktiviteter, statusuppdateringar, som rör datamängder i SmartFlow. Dessa aktiviteter refereras till som händelser senare i kapitlet.

Som grund för att beskriva allt detta i en RDF miljö har vi använt DCAT med profilen GeoDCAT-AP.

DCAT (en W3C-standard) är en RDF-vokabulär för att representera datakataloger. GeoDCAT-AP är en utökning av DCAT-AP för att beskriva geospatiala datamängder och datatjänster. DCAT-AP i sin tur är en utökning av DCAT för att beskriva dataportaler i Europa.

För en beskrivning av begrepp som projekt, skede och leveransspecifikation, se [1].

Se kapitel 5.1 för en modell över GeoDCAT-AP version 2.

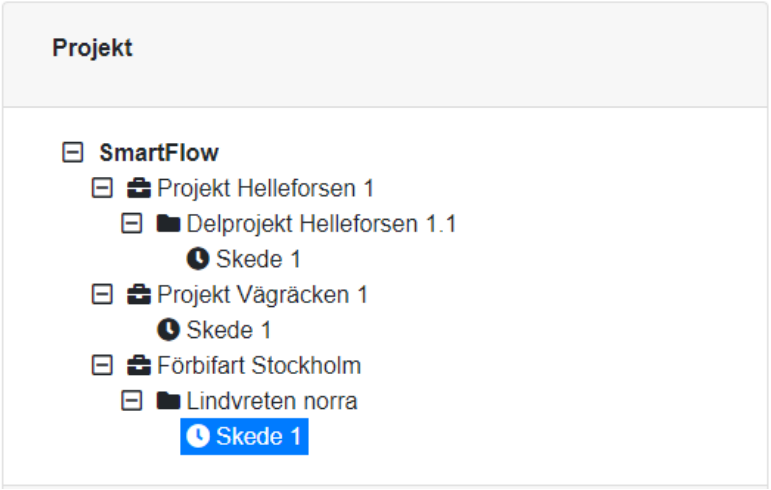
Se kapitel 5.2 för ett modell-exempel på hur vi har använt GeoDCAT för att representera projektstruktur, leveransspecifikationer, datamängder och händelser.

Struktur

För att representera en nod (projekt, delprojekt, skede) i en projekt-struktur har vi använt klassen dcat:Catalog. En dcat:Catalog-instans kan ha flera dcat:Catalog-instanser som underkataloger vilket möjliggör en hierarki av kataloger. En tänkbar hierarki är ”projekt har delprojekt som i sin tur kan ha delprojekt som kan ha skeden”. En grundare hierarki skulle kunna innebära att ”projekt har skeden”.

I Figur 2 finns ett exempel på projekt-struktur som utgått från tre testfall i AP2. Överst i bilden finns en huvudkatalog som representerar SmartFlow. I den katalogen finns tre registrerade projekt, ett per testfall i AP2. I två av projekten finns fiktiva delprojekt. I delprojekten och i ett av projekten finns fiktiva skeden registrerade. Alla dessa noder representeras av varsin dcat:Catalog-instans.

Hur man väljer att definiera huvudkatalogen (i exemplet SmartFlow) kan bero på sammanhanget. Kanske representerar den en hel organisation, kanske representerar den ett stort projekt.



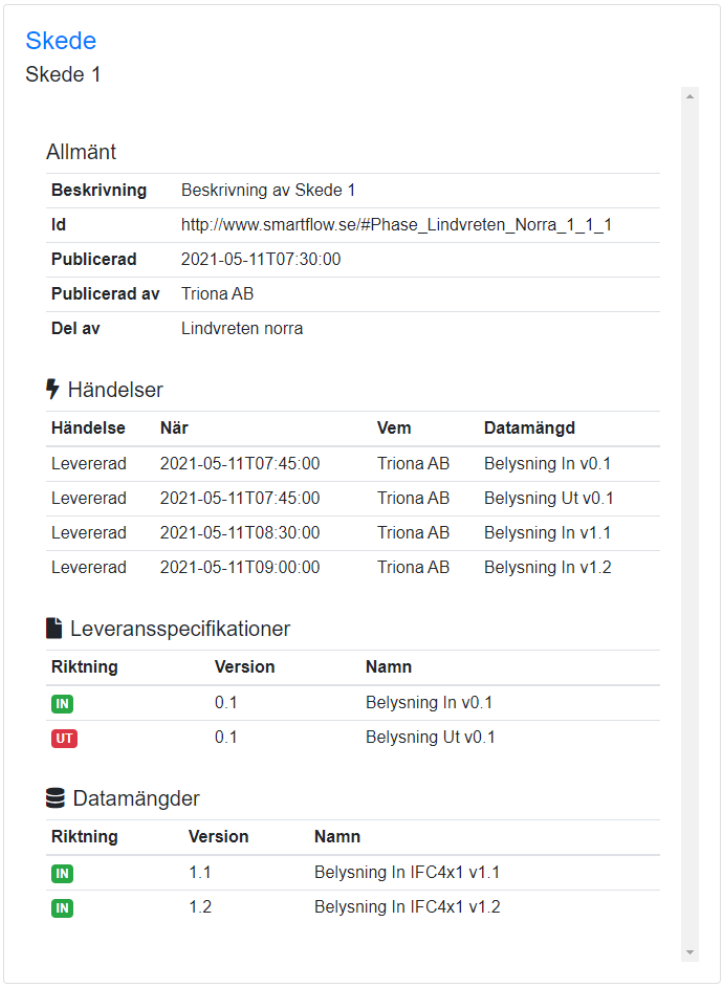
Figur 2 – Exempel projektstruktur med dess delprojekt och skeden

Förutom sin plats i katalog-hierarkin kan en katalog ha en mängd information kopplat till sig, bl.a. information om när och av vem den publicerades, titel och en beskrivning av ex. innehållet.

Det primära innehållet i en katalog är datamängder. Teoretiskt kan alla kataloger i en hierarki innehålla datamängder men i vårt exempel i Figur 2 är det löven i en gren, dvs. skeden, som innehåller datamängder. Datamängderna i en katalog är levererade indatamängder och utdatamängder (dvs transformerade indatmängder) för konsumtion.

Datamängderna representeras av instanser av klassen dcat:Dataset

I Figur 3 visas ett exempel med information och innehåll för Skede 1.



Figur 3 - Information om och innehåll i Skede 1, Lindvreten norra

Leveransspecifikationer

I AP2 har vi valt att representera en leveransspecifikation med en instans av klassen dcat:Dataset, precis som för in- och utdatamängder. Leveransspecifikationen representerar dock inte en faktisk datamängd i SmartFlow utan beskriver en förväntad datamängd och vad som förväntas av den.

I och med att en leveransspecifikation är ett dcat:Dataset är den också en del av innehållet i en katalog. Dvs. man registrerar en leveransspecifikation i den katalog där specifikationens förväntade datamängd också kommer att registreras.

Ett dcat:Dataset, och därigenom en leveransspecifikation, kan finnas i flera versioner. Vi har valt att versionsnumrera leveransspecifikationen enligt mönstret 0.X. Första versionen av en leveransspecifikation får version 0.1. Om den behöver uppdateras representeras den nya versionen av en ny instans av klassen dcat:Dataset med versionsnummer 0.2 osv. dcat:Dataset gör det möjligt att etablera versionsrelationer mellan leveransspecifikationer så att vi vet om en leveransspecifikation har en nyare version eller är en nyare version av en äldre leveransspecifikation.

Leveransspecifikationer kan registreras för både indatamängder (inleverans) och utdatamängder (utleverans/konsumtion). dcat:Dataset gör det möjligt att skapa en relation mellan en

utleveransspecifikation och en inleveransspecifikation så att vi kan säga att en förväntad utdatamängd som ska uppfylla en viss utleveransspecifikation är resultatet av en transformation av en indatamängd som ska uppfylla en viss inleveransspecifikation.

Då både leveransspecifikationer och datamängder representeras av instanser av dcat:Dataset kommer de att delvis ha samma egenskaper. Skillnaden ligger då i tolkningen. Ett exempel är publicist; för leveransspecifikationen är tolkningen ”den förväntade indatamängden ska levereras av X” men för datamängden är tolkningen ”datamängden har levererats av X”. Samma fenomen gäller andra egenskaper, till exempel format, koordinatsystem och geografisk utbredning.

Datamängder

Som sagts tidigare beskrivs datamängder (levererade indatamängder och utdatamängder för konsumtion) med klassen dcat:Dataset. Den faktiska, fysiska, datamängden ses som en distribution och representeras av klassen dcat:Distribution vilken innehåller någon form av pekare till var data lagras.

Ett exempel. En datamängd i form av en IFC 3D-modell levereras till SmartFlow under ett visst skede. Data konverteras till RDF och lagras i en graf i GraphDB. 3D-modellen läses också in i Quadri Server för att kunna visualiseras där. För att enkelt kunna titta på de levererade företeelsernas geografiska läge konverteras och lagras datamängden också i en geografisk relationsdatabas med tillhörande karttjänst. Datamängden beskrivs i SmartFlow med en instans av dcat:Dataset vilken har relationer till fyra instanser av dcat:Distribution:

1. En distribution som beskriver och pekar på indatafilen (IFC 3D-modell)
2. En distribution som beskriver och pekar på grafen där data för datamängden är lagrat i GraphDB
3. En distribution som beskriver och pekar på datamängden i Quadri Server
4. En distribution som beskriver och pekar på karttjänsten för datamängden i relationsdatabasen

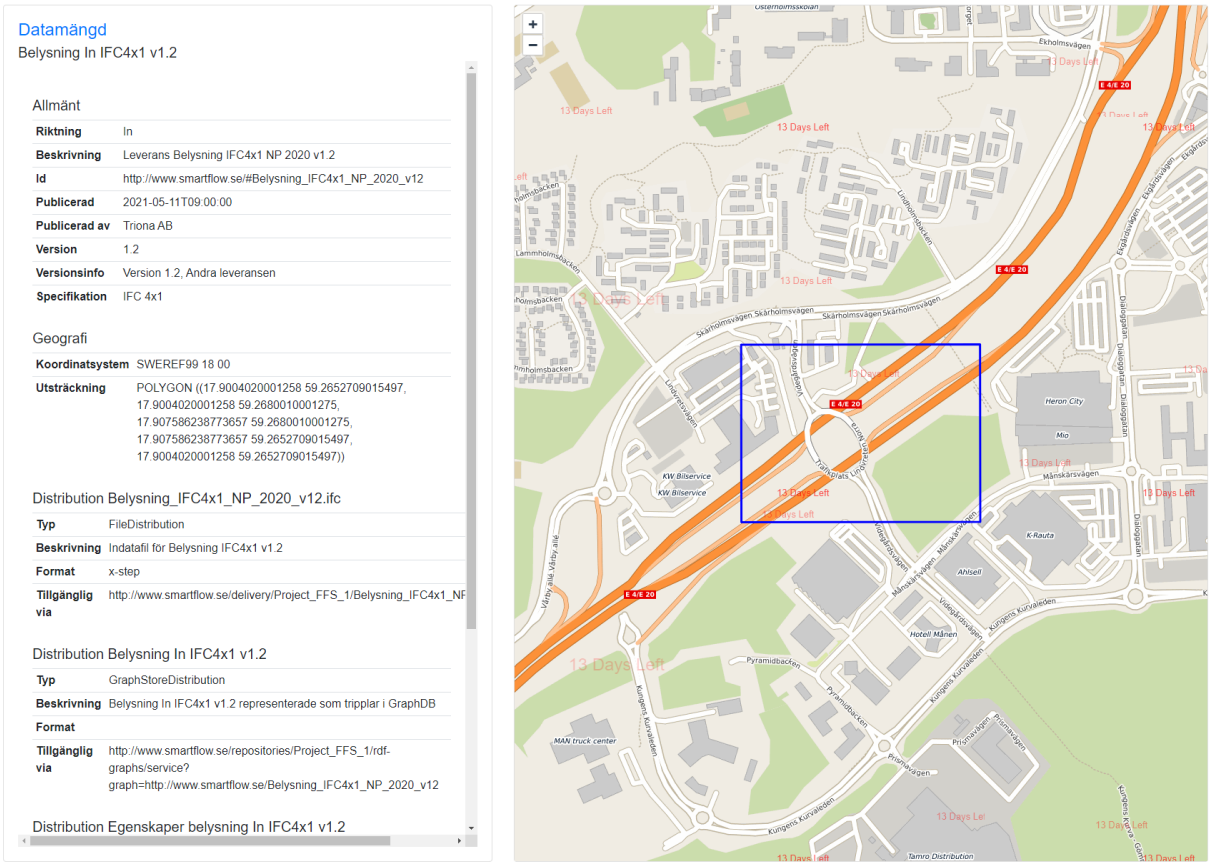
Indatamängden transformeras sedan enligt en utleveransspecifikation och lagras som RDF i en graf i GraphDB. Utleveransspecifikationen talar om i vilken form utdatamängden ska konsumeras. I detta exempel kan vi säga att utdata bara ska konsumera som en OpenTNF/Geopackage-fil. SmartFlow konverterar därför utdatamängden i RDF till en OpenTNF/Geopackage-fil. Utdatamängden beskrivs av en instans av dcat:Dataset vilken har relationer till två instanser av dcat:Distribution:

1. En distribution som beskriver och pekar på grafen där data för datamängden är lagrat i GraphDB
2. En distribution som beskriver och pekar på den publicerade filen i OpenTNF/Geopackage-format

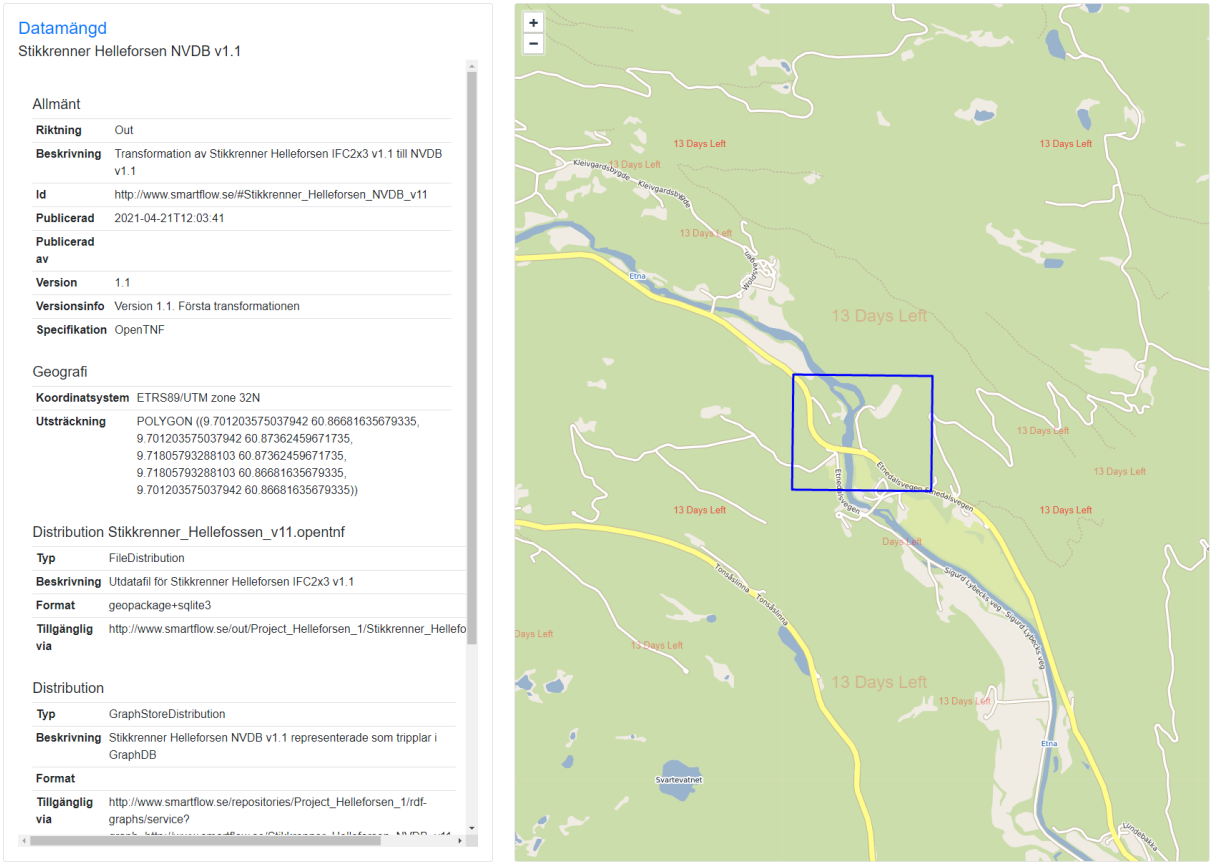
Precis som för leveransspecifikationer kan datamängder versionshanteras med hjälp av DCAT. I AP2 har vi sett en datamängd som en version av den leveransspecifikation den uppfyller vilket ger spårbarhet mellan datamängd och specifikation. Versionsnumreringen följer mönstret X.Y där X är versionen på den leveransspecifikation datamängden uppfyller och Y är versionen på datamängden. Om exempelvis en datamängd, som ska uppfylla en specifikation version 0.1, levereras till SmartFlow för första gången får den version 1.1. Om en uppdaterad version av datamängden levereras får den versionen 1.2. Om specifikationen uppdateras med en ny version 0.2 och det kommer in en ytterligare uppdaterad leverans av datamängden men som nu uppfyller specifikation 0.2, får datamängden version 2.1, dvs första versionen mot specifikation 0.2 och är också en version av sin leveransspecifikation.

På samma sätt som för leveransspecifikationer kan vi med hjälp av DCAT också skapa en relation mellan en utdatamängd och den indatamängd den är transformerad från.

Se Figur 4 och Figur 5 för ytterligare exempel på beskrivningar av in- och utdatamängder.



Figur 4 - Information om den levererade indatamängden Belysning IFC4x3 v1.2



Figur 5 - Information om den transformerade utdatamängden Stikkrenner Hellefossen NVDB v1.1

Händelser

Det finns ett behov av att beskriva status eller händelser för de aktiviteter som ingår i flödet av data genom SmartFlow.

I AP2 har vi valt att representera en sådan händelse/statusuppdatering med hjälp av klassen dcat:CatalogRecord.

En händelse på en datamängd i en katalog representeras av en instans av klassen dcat:CatalogRecord. Varje ny händelse på en datamängd blir en ny instans. Därigenom utgör unionen av alla instanser av dcat:CatalogRecord en händelselogg över händelser för datamängder i SmartFlow.

I Figur 3 visas ett exempel där fyra händelser har registrerats för Skede 1 där man kan se att datamängder har levererats till SmartFlow under Skede 1 en viss tid av Triona AB.

2.2.4 CoClass

För arbetspaket 3 tänker vi oss att använda CoClass. För att på enklaste sätt använda CoClass i en miljö för länkade data så tänker vi ladda ner CoClass innehåll via CoClass api och konvertera data till OWL på följande sätt:

- Varje Klass i CoClass blir en owl:Class
- Subklasser i CoClass definieras som subklasser i owl med rdfs:subClassOf
- Varje Egenskap i CoClass blir en owl:DatatypeProperty (eller owl:ObjectProperty)

- Sub-egenskaper i CoClass definieras som sub-properties i owl med rdfs:subPropertyOf
- Vi definierar en URI för varje klass och egenskap samt knyter till övriga egenskaper (t ex namn och definition) till respektive definition.

Inom ramen för arbetspaket 2 har vi skaffat api-nycklar till CoClass så att arbete kan starta direkt i arbetspaket 3.

En arbetshypotes är att CoClass kan utgöra en ”kanonisk” struktur dit alla indata knyts automatiskt via länkade ontologier och semantisk reasoning. Detta medger att generering av utdata mycket enklare kan göras baserat på EN nomenklatur (CoClass) snarare än diverse olika strukturer, versioner och dialekter.

2.2.5 Övrigt

Inom arbetspaket 2 har vi startat ett arbete med att detaljera kravbilden när det gäller processer och flöden. Detta berör frågeställningar som:

- Metadatastrukturer i SmartFlow
- Vilka behov finns för att definiera strukturer i SmartFlow, t ex projekt, delprojekt, skeden, leveranser, rättigheter för att läsa och skriva etc
- När definieras dessa strukturer?
- Vilka data levereras när?
- Hur kravställs data?
- Hur och när verifieras data mot krav?

Detta arbete förväntas fortsätta under hela arbetspaket 3.

Vi har även startat ett arbete när det gäller kvalitetsattribut (icke-funktionella krav) för lösningen. Detta kan t ex röra saker som prestanda, skalbarhet etc. För detta syfte har ett frågeformulär skickats till behovsägarna som besvarat detta. Arbete med att omsätta resultatet i en arkitektur ska bedrivas under arbetspaket 3.

Eftersom vi inte har någon djup erfarenhet av praktisk användning av triple stores och GraphDB i synnerhet så behöver vi skaffa oss en bild av hur produkten bäst kan användas för att hantera de data och metadata som vi ser framför oss i projektet. Lärdomar angående detta beskrivs i kapitel 2.3.1.

2.3 Lärdomar

2.3.1 Verktyg för Länkade data/Semantisk webb

Bakgrund

Den miljö som används för lagring och bearbetning av RDF-data i SmartFlow är, som tidigare nämnts, GraphDB. GraphDB har stöd för lagring av RDF-data, vilket innebär både ontologier (schema/ontologi eller TBOX) och instanser (ABOX). Tillsammans brukar man kalla detta för kunskapsgrafer. Utifrån befintliga RDF-data (assertions), kan miljön via så kallad reasoning härleda nya data (inferred data). Rent praktiskt fungerar det så att miljön utifrån fördefinierade regler tolkar befintliga data (i form av tripplar) och härleder nya tripplar utifrån dessa.

Ett enkelt exempel kan vara:

Assertions:

:myObject_1 rdf:type ifc:IfcWall #Ett objekt som är av typen IfcWall

ifc:IfcWall owl:equivalentClass CoClass:B #Klassen IfcWall är ekvivalent med klassen B i CoClass

Härledda data:

myObject_1 rdf:type CoClass:B #Objektet myObject_1 tillhör klassen CoClass:B

På detta sätt kan man genom att när som helst lägga in assertions, t ex att IfcWall och CoClass:B beskriver samma mängd/klass, få miljön att automatiskt klassificera data genom att på schema-nivå lägga till information om hur olika begrepp förhåller sig till varandra.

Som frågespråk har GraphDB stöd för SPARQL (inklusive GeoSPARQL) som medger även möjlighet till spatiala utsökningar/frågor. GraphDB har även stöd för GraphQL.

Utöver detta har GraphDB stöd för SHACL (https://www.w3.org/TR/shacl/) som kan användas för att verifiera en mängd tripplar mot krav (t ex att ett objekt måste ha ett visst attribut och att attributet måste vara ett heltal mellan 0 och 100). SHACL-regler beskrivs också i form av RDF-grafer och kan precis som scheman och data lagras i GraphDB. Där det finns regler kommer GraphDB att vid lagring verifiera data mot reglerna och förhindra lagring vid ”regelbrott”.

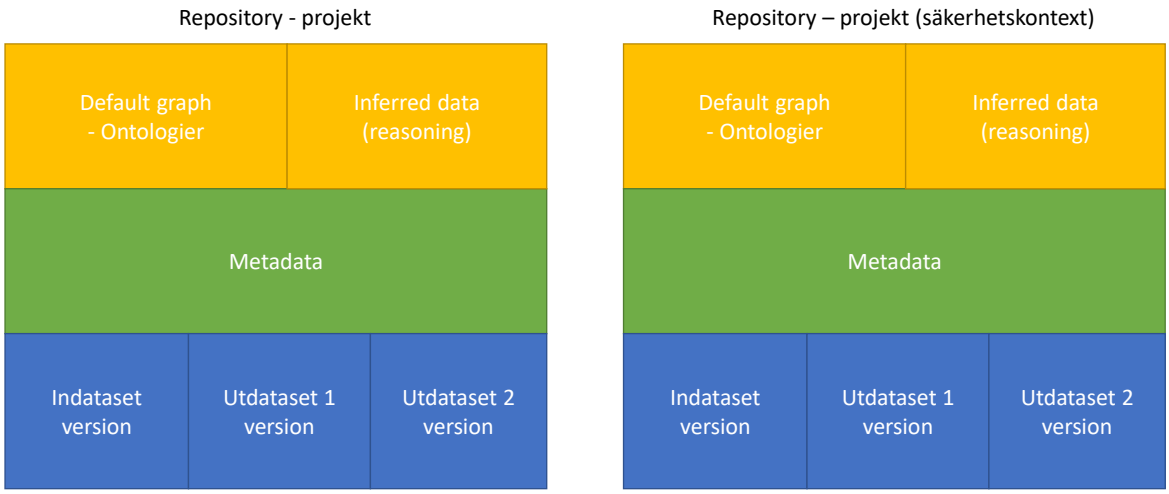
GraphDB har även stöd för import/konvertering av diverse format via ”OntoRefine”. Via denna funktion kan man importera och skapa tripplar från exempelvis databastabeller, excel-filer, xml-filer, json-filer etcetera. Dessa data kan sedan, som visats i testfallet med belysningsstolpar enkelt länkas mot andra data under förutsättning att det finns identitetsbegrepp som kan användas för att förstå vad som ska länkas med vad.

En GraphDB-instans kan hantera flera repositories (databaser) där varje repository kan innehålla grafer. Varje graf innehåller en mängd tripplar. De data som graphDB härleder via reasoning läggs i en separat fördefinierad graf. Under förutsättning att man har rättigheter så går det att ställa frågor tvärs repositories och naturligtvis även tvärs grafer i ett repository.

Rättigheter sätts per repository.

Struktur i GraphDB

Vi har kommit fram till att vi vill använda följande grundläggande struktur i GraphDB



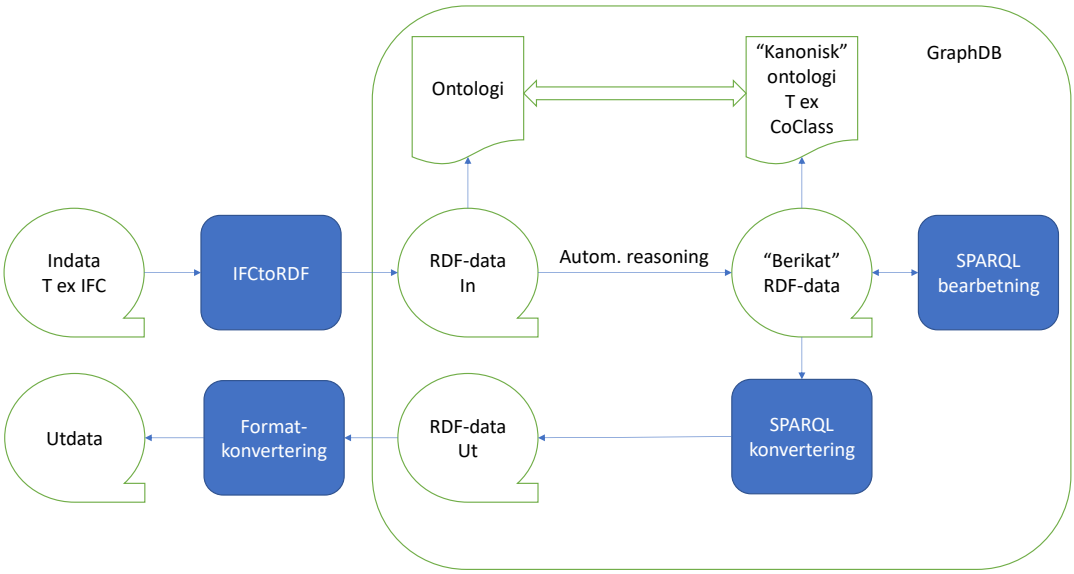
Figur 6 - Struktur i GraphDB

Ett projekt motsvarar ett repository. Om projektet har känsliga data får dessa läggas i separata repositories med separata inställningar för tillgång.

- Ontologier läggs i repositoryt:s default-graf.
- De data som GraphDB själv kan härleda hamnar automatiskt i den graf som av GraphDB definierats för detta
- Metadata läggs i separat graf. Metadata gäller hela repositoryt
- Varje importerad fil läggs i separat graf. Varje graf motsvarar en version av filen (i metadata hålls flera versioner av samma dataset samman)
- De data som genereras av SmartFlow för konsumtion läggs i separata grafer som via metadata kopplas som separata distributioner av datasetet.

Bearbetning i GraphDB

För att få kontroll på vilka data som hör till vad så tänker vi oss att man bör hantera data på följande sätt:



Figur 7 – Bearbetningskedja

1. Indata konverteras till RDF
2. Vid import av RDF-data klassas data automatiskt (primärt via reasoning) mot en ”kanonisk” ontologi. Denna kan exempelvis vara CoClass eller något annat som passar användningen
    a. Detta medger att efterföljande utsökningar blir oberoende av indataformatets vokabulär (t ex kan det skilja mellan IFC2x3 och IFC4)
    b. Mappningen sker med speciella ontologier som länkar indataontologin mot den kanoniska ontologin.
3. Viss berikning av data kan ske ytterligare via specifika SPARQL-scripts. Detta kan t ex vara att generera GeoSPARQL-geometrier (punkt, linje, yta) för objekten så att dessa kan visas på karta, sökas ut spatialt samt matchas mot nät.
4. Specifika utdata görs via SPARQL-scripts där man har alla möjligheter att göra beräkningar, aggregeringar osv för att anpassa utdata mot specifika krav
5. Till sist kan utdata behöva konverteras till nytt format för leverans

Geometriförenkling

IFC har en relativt komplex struktur för att beskriva 3D-geometrier med många relationer innan man från ett objekt kommer till själva geometrin. Dessutom är den vanligaste formen för geometrisk representation av 3D-objekt någon form av BREP-geometri (ofta IfcFacetedBrep), där en 3D-geometri beskrivs av triangulära facetter med många gånger hundratusentals facetter och ännu fler punkter. Det sätt som IfcOWL konstruerats på medger en exakt representation av IFC-strukturen för att möjliggöra tvåvägskommunikation. Detta innebär att en IFC-fil i värsta fall kan innehålla miljontals tripplar bara för att beskriva geometri. Redan vid ett tidigt stadium var rekommendationen från Ontotext att titta på sätt att i den semantiska miljön fokusera på semantik och att hantera detaljerad 3D-geometri i en miljö specialiserad för detta.

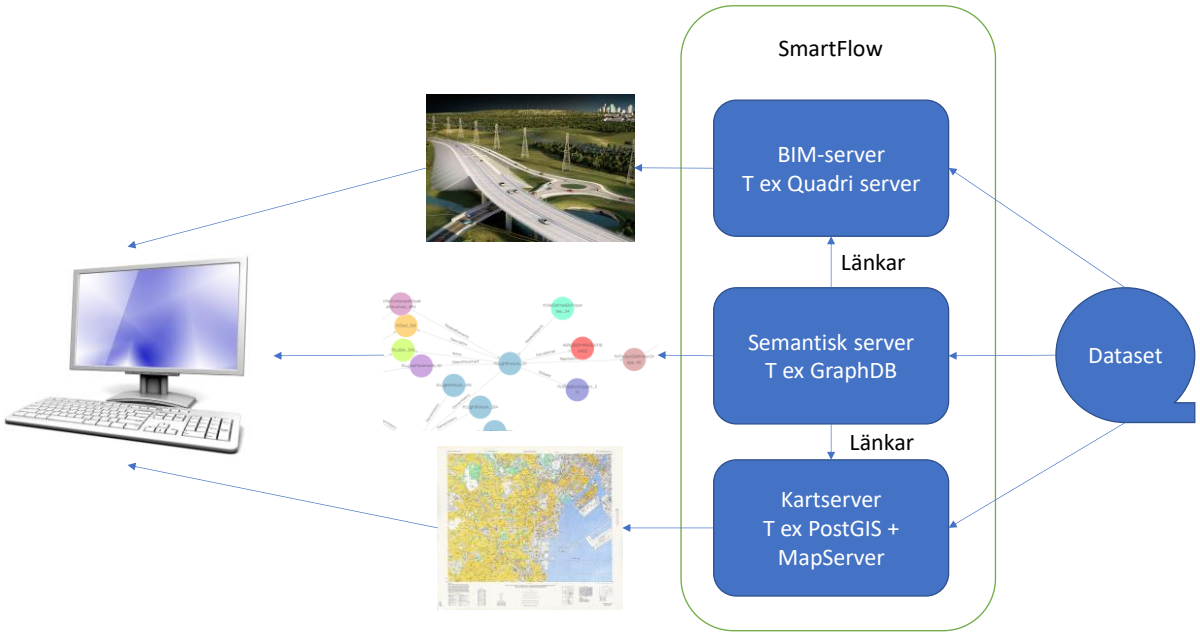
Vi har i AP 2 tittat på detta på flera sätt:

- Trimble har levererat flera versioner av IFC-filer där en version innehåller 3D-geometri och en version innehåller enklare geometri (t ex polylines)
- Vi har med hjälp av SPARQL genererat GeoSPARQL-geometri och därigenom gjort förenkling inom ramen för GraphDB-miljön.

Antagligen är den bästa approachen att göra mesta möjliga förenkling innan data läses in i GraphDB. Om en fil med miljontals tripplar läses in är risken att avancerad reasoning görs helt i onödan vilket medför sämre prestanda.

Kart- och BIM-servrar

Baserat på ovanstående så kan man tänka sig att sätta upp en miljö enligt nedan för att uppnå maximal funktionalitet och prestanda:



Figur 8 - Tänkbar miljö för maximal funktionalitet och prestanda

3 Förslag till prioriterade aktiviteter i arbetspaket 3

3.1 Utveckla och beskriv arkitektur

3.1.1 Förfina use cases/user stories

I AP1 definierades ett antal övergripande användningsfall för SmartFlow. I AP3 finns ett behov av att se över och detaljera användningsfallen tillsammans med projektets intressenter för att ta reda på vilka funktionella krav som ställs på ett system som SmartFlow och därigenom kunna bygga ett relevant demo-system. Se kapitel 2.2.5 för exempel på frågeställningar som är intressanta i detta arbete.

Det innebär bl.a. att noggrannare titta på processerna för inleverans av data till SmartFlow och konsumtion av data från SmartFlow, vem som gör vad och när samt detaljera behoven av stödjande användningsfall som exempelvis administration och visualisering av data.

Hur användningsfallen ska prioriteras när demo-systemet ska konstrueras behandlas översiktligt i kapitel 3.1.8.

3.1.2 Definiera egenskapskrav (icke funktionella krav)

För att kunna beskriva en tänkbar arkitektur för ett system som SmartFlow är det viktigt att veta vilka förväntningar som finns på systemets beteende. Dessa kan uttryckas som t.ex. egenskapskrav eller ”Vad händer om”-scenarion, även kallade stressorer. Områden där det finns förväntningar kan handla om prestanda, datamängder, samtidighet, förändringsbarhet, förvaltningsbarhet, säkerhet, etc.

Det är vanligt att det finns implicita förväntningar redan från början, baserade på verksamhetskunskaper och erfarenhet baserad på konstruktion av andra system, som styr den initiala idéen om arkitekturen.

Det vi ska göra i AP3 är att explicit beskriva förväntningar på systemets beteende i någon form på en rimlig nivå för att kunna leverera en beskrivning av en arkitektur och kunna implementera hela eller delar av den i ett demo-system

Arbetet med detta är redan påbörjat, som uppgetts tidigare i dokumentet, då ett frågeformulär skickats till behovsägarna som besvarat detta.

3.1.3 Principer för lagring av data och metadata i triple store

Det som beskrivits i kapitel 2.3.1 angående principer för lagring i GraphDB behöver förfinas och beskrivas/specificeras. Särskilt behöver frågor kring processer kring indataleveranser samt även eventuella säkerhetsaspekter belysas.

3.1.4 Principer för datatransformationer

I kapitel 2.3.1 beskriver vi några tankar kring datatransformationer och hur dessa ska utföras.

Tanken är att indata i möjligaste mån vid import ska klassificeras mot en ”kanonisk” ontologi som för Sverige skulle kunna vara CoClass och för Norge exempelvis SOSI eller nvdb-owl. Detta bör gälla såväl klasser som egenskaper. Om detta görs kan efterföljande arbete med mappning mot konsumerande system göras mot denna kanoniska ontologi istället för unikt mot varje form av indata.

Dessa principer bör slås fast och testas.

När det gäller IFC, kommer de flesta egenskaperna att komma via property sets som är en generell struktur av listor med egenskaper som beskrivs enligt principen ”tag/value”. Denna struktur lämpar sig dåligt för automatisk omklassning via reasoning. I dessa tillfällen kan det vara klokt att om möjligt använda egenskapsdefinitioner direkt från den kanoniska modellen för att förenkla mappning.

3.1.5 Komponenter/GUI

En del i att ta fram en arkitektur är att skapa en eller flera modeller som beskriver vilka komponenter som systemet byggs upp av och vilket/vilka ansvar varje komponent har. Komponentiseringen drivs bl.a. av de egenskapskrav som ställs på systemet.

Arkitekturen beskrivs troligen med olika komponenter beroende på perspektiv, ex. funktion eller ”run-time”.

Summa summarum, i AP3 ska ta vi fram en arkitektur där vi paketerar implementeringen av de kravställda funktionerna så att vi uppfyller egenskapskraven. Det inkluderar klienter med användargränssnitt, 3D-tjänster, karttjänster, komponenter för att transformera och konvertera data, komponenter som sköter orkestrering av flöden, databashanterare, köer m.m.

3.1.6 Deployment

Som en del av framtagandet av egenskapskrav ska vi i AP3 undersöka behov och konsekvenser av olika val av exekveringsmiljöer för systemet SmartFlow och hur vi arkitekturellt kan hantera det. Vad händer om man vill köra SmartFlow i molnet? Vad behöver göras för att köra ”On-Premise”? Vad innebär olika val av komponenter och externa system för att tillgodose behovet av funktioner (GraphDB, Quadri Server, TNE, GeoServer/MapServer etc)?

3.1.7 Kart- och 3D-tjänster

Redan under AP1 har det framkommit idéer om att kunna visualisera 3D-representationen av ett objekt genom att eventuellt välja objektet i SmartFlow eller kanske hela den 3D-modell som levererats till SmartFlow där objektet ingår. Hur detta kan gå till ska undersökas och testas i AP3.

Ett annat uttryckt förväntat behov är möjligheten att visualisera det geografiska läget för objekt som levererats till SmartFlow. Under AP2 har detta testats genom att läsa upp objektens geometrier (i GeoSparql-form) från GraphDB, konvertera dem till det format som lämpade sig för den kartkomponent som användes i demo-applikationen i AP2 och rita ut dem. Det finns idéer om att det skulle vara effektivare att använda karttjänster för att visualisera objektens geometrier. En positiv bieffekt med en sådan infrastruktur på plats är den relativt lilla insatsen det innebär att erbjuda system att konsumera data från SmartFlow i form av geodatatjänster som WFS. Detta ska undersökas och testas i AP3.

3.1.8 Definiera scope for demonstrationer

Projektet ska demonstreras på ett sätt som tydliggör nyttor med SmartFlow-ansatsen för experter inom både IT och infrastruktur. Det är därför viktigt att arbetspaket 3 tidigt definierar ett tänkbart scope för att kunna demonstrera lösningen.

3.2 Utveckla lösning inkl GUI för demonstrationer (inklusive 3D-visualisering t ex med Quadri och/eller karttjänster)

För att testa och demonstrera de hypoteser som tas fram under arbetet i SmartFlow-projektet ska vi under AP3 utveckla en lösning som bygger på hela eller delar av den framtagna arkitekturen och som implementerar alla eller vissa av de funktioner som definierats i framtagna användningsfall.

Vad som ska implementeras relaterar till det scope som definieras i kapitel 3.1.8.

Nedan följer en beskrivning av några explicita områden som vi tror kommer att ingå i det test- och demonstrations-scope som definieras.

3.2.1 Utveckla lösning för geometriförenkling

Utifrån hypotesen att komplexa 3D-modeller inte ska lagras i RDF-format i SmartFlow vill vi testa en lösning som innebär att SmartFlow med automatik kan förenkla 3D-representationerna av objekt till enkla geografiska läges-representationer och då troligen tidigt i SmartFlow-systemets inleveransprocess.

3.2.2 Utveckla mappning till topologiskt nät

Det finns scenarion där objekten i en inleverans saknar koppling till ett topologiskt nät men där ett system som vill konsumera det transformerade resultatet av den inleveransen har ett behov av att få med objektens utbredning längs ett topologiskt nät.

Vi vill testa en lösning som innebär att exempelvis TNE exponerar funktionen mappning till topologiskt nät och användas med automatik av SmartFlow på lämpligt ställe i processen för inleverans/transformation/utleverans.

3.2.3 Hantering av koordinatsystem/transformationer

Geometrier i en till SmartFlow levererad datamängd beskrivs (förhoppningsvis) i ett definierat koordinatsystem. Ett system som ska konsumera datamängden vill att geometrierna beskrivs i ett annat koordinatsystem. SmartFlow behöver ha stöd för flera olika koordinatsystem och transformationer. Dessutom vill vi på ett smidigt sätt kunna visualisera geografiska lägen för objekt levererade till SmartFlow oavsett vilket koordinatsystem dess geometrier har.

Vi vill testa en lösning som hanterar fallet med olika definierade koordinatsystem in och automatisk transformering till ett annat koordinatsystem ut där ex. risken för tappad noggrannhet och/eller precision är hanterbar. Samtidigt som en visualisering i SmartFlow av objektens geografiska läge eller söka efter objekt i SmartFlow baserat på deras geografiska läge ska vara möjligt.

3.3 Ta fram representativa testfall

För att kunna utföra nyttiga demonstrationer samt att verifiera och validera lösningen så tror vi att det är viktigt att vi gör detta utifrån några definierade och relevanta testfall. Detta behöver specificeras i ett tidigt skede av AP 3.

För detta så behöver ett antal saker finnas på plats:

- Representativa dataset
    - Dessa dataset behöver täcka såväl ”BIM-data” från investeringsprojekt som data från inventeringar
- Vi behöver definiera några konsumenter av ovan nämnda dataset samt definiera deras informationskrav
- Vi behöver definiera en relevant process för vart och ett av dessa testfall som ska kunna demonstreras på ett tydligt sätt

3.4 Användning av CoClass

I kapitel 2.2.4 beskriver vi våra tankar runt CoClass.

Rent konkret så behöver ett antal punkter genomföras:

- Kod behöver skrivas som via CoClass API laddar ner CoClass och genererar en ontologi som motsvarar CoClass struktur för klasser och egenskaper
- Vi behöver även utreda hur vi skulle kunna ta hand om eventuell kravställning i form av Trafikverksspecifika strukturer (som beskrivits i CoClass studio) för att tex generera SHACL-restriktioner för verifiering av data
- Vi behöver testa huruvida CoClass kan användas som ”kanonisk” struktur dit all indata kan mappas i ett första steg
- Vi behöver antagligen också hantera de fall där CoClass-klassifikation använts direkt i IFC-data

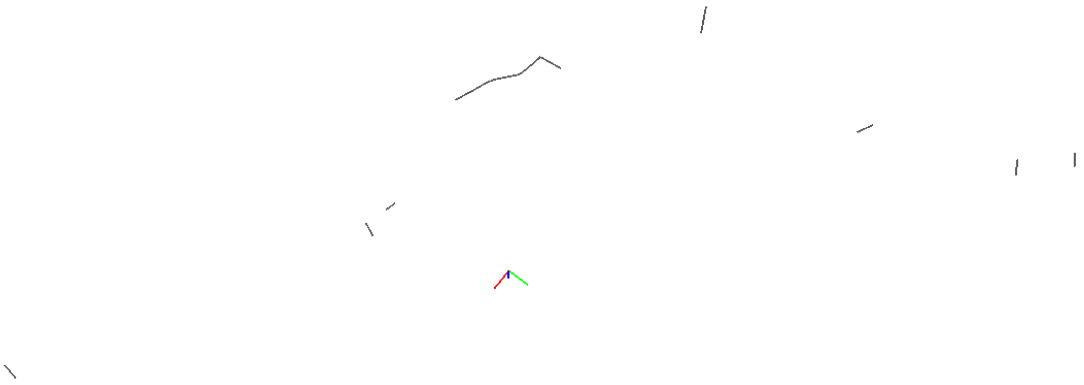
3.5 Definition och användning av ”kanonisk modell” för Statens Vegvesen

För Trafikverket tänker vi oss att använda CoClass som en ”kanonisk modell” för att initialt kunna mappa data till en enhetlig struktur. För Statens Vegvesen eller någon annan som inte använder CoClass så bör man kunna använda samma princip men med andra ontologier som grund. Därför vill vi identifiera någonting för Statens vegvesen som fyller samma funktion som CoClass gör i Sverige, exempelvis SOSI eller nvdb-owl. Detta behöver definieras och testas.

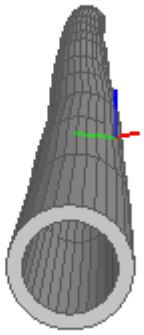
4 Appendix 1 – Beskrivning av testfall

4.1 Stikkrenner hellefossen – IFC (Statens Vegvesen/Trimble)

Från Statens vegvesen och Trimble har vi fått data på IFC-format (IFC 2x3 och IFC4) som beskriver Stikkrenner/Vägtrummor (se figurer).



Figur 9 - Stikkrenner Hellefossen översikt



Figur 10 - Exempel på Stikkrenne

För att underlätta hanteringen så bistod Trimble med att generera två versioner av IFC-filerna, en med 3D-geometri och en med en enklare 1D-geometri (linjer).

Processen för hantering av data gick till på följande sätt:

0. Import av berörda ontologier till default-graf
    a. IfcOWL
    b. Nvdb-owl
    c. DCAT (för metadata)
1. Konvertering av IFC till RDF (Turtle) med hjälp av IFCtoRDF
2. Import av RDF till GraphDB till separat graf
3. Generering av metadata (semi-automatiskt)
4. Datakonvertering till nvdb-owl med hjälp av SPARQL
    a. Med SPARQL skapas tripplar som läggs till med IFC-objekten som subjekt, dvs inga nya URI:er skapas
    b. IFC:s sätt att hantera property sets (en lista av tag/value) gör att man behöver använda SPARQL för att tolka/konvertera attribut
    c. För värden ur kodlistor matchas indatasträngar mot rdfs:label för individer i kodlista
    d. För literalvärden så kan värdet direkt kopieras och knytas mot rätt predikat från IfcOWL
    e. SPARQL för själva konverteringen finns i kapitel 7.1
5. Resultatet av datakonverteringen läggs i egen separat graf
6. Skapa metadata för utdata
7. Export till excel
    a. Via en enkel SPARQL-select och export av svaret till csv-format. Nedan ses exempel från fyra st objekt (tabellen uppdelad för läsbarhet). Selektionen som ger nedanstående resultat finns i kapitel 7.2

uri	guid	mtrl	mtrlNavn
inst:IfcFlowSegment_112	1ffa4lFKTFM8JFw7Mern7R	http://rdf.vegdata.no/nvdb/nvdb-owl#tv9125	Betong
inst:IfcFlowSegment_137	2O2zDZsnP5$gqYUN4CVX$G	http://rdf.vegdata.no/nvdb/nvdb-owl#tv9125	Betong
inst:IfcFlowSegment_156	0DffXUa3v1XPESY9OmAERd	http://rdf.vegdata.no/nvdb/nvdb-owl#tv9127	Plast
inst:IfcFlowSegment_174	32JSXSWl11meQvlqGEz54x	http://rdf.vegdata.no/nvdb/nvdb-owl#tv9125	Betong

innType	innTypeNavn	bruk
http://rdf.vegdata.no/nvdb/nvdb-owl#tv11744	Åpent i grøft	http://rdf.vegdata.no/nvdb/nvdb-owl#tv9114
http://rdf.vegdata.no/nvdb/nvdb-owl#tv11744	Åpent i grøft	http://rdf.vegdata.no/nvdb/nvdb-owl#tv9114
http://rdf.vegdata.no/nvdb/nvdb-owl#tv11744	Åpent i grøft	http://rdf.vegdata.no/nvdb/nvdb-owl#tv9114
http://rdf.vegdata.no/nvdb/nvdb-owl#tv11744	Åpent i grøft	http://rdf.vegdata.no/nvdb/nvdb-owl#tv9114

brukNavn	diam	aar	geom
Vann	400	2018	<http://www.opengis.net/def/crs/EPSG/0/3006> MULTILINESTRING((538549.67 6748418.88 251.78,538549.39 6748418.27 251.72,538548.66 6748416.81 251.59,538547.6800000001 6748414.81 251.45,538546.6899999999 6748412.82 251.3,538544.6899999999 6748408.85 250.97,538543.6800000001 6748406.88 250.85,538543.0600000001 6748405.64 250.72,538542.77 6748405.04 250.68))
Vann	600	2018	<http://www.opengis.net/def/crs/EPSG/0/3006> MULTILINESTRING((538133.61 6748875.85 264.83,538132.91 6748875.21 264.84,538122.16 6748865.27 264.98))
Vann	300	2018	<http://www.opengis.net/def/crs/EPSG/0/3006> MULTILINESTRING((538538.54 6748442.62 252.27,538543.6899999999 6748426.61 252.09))
Vann	600	2018	<http://www.opengis.net/def/crs/EPSG/0/3006> MULTILINESTRING((538083.26 6748904.77 265.07,538083.87 6748905.48 265.07,538084.46 6748906.19 265.06,538085.89 6748907.91 265.05,538087.29 6748909.66 265.04,538088.6899999999 6748911.4 265.03,538090.1 6748913.13 265.02,538090.85 6748914.08 265.02,538091.47 6748914.83 265.02))

4.2 Belysningsstolpar – IFC (Trafikverket/Trimble)

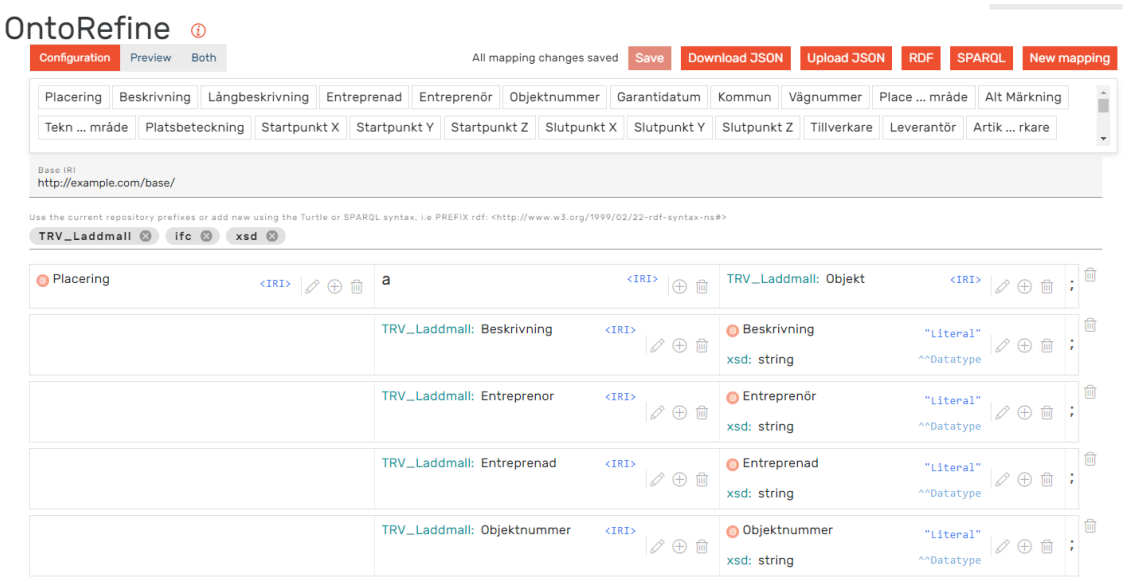
Belysningsstolpar levererades på IFC4-format i två omgångar:

- Omgång 1
    - Belysningsstolpar med 3D-geometri (IfcFacetedBrep)
    - Unika komponent-id för stolpar i property set
    - Ej unika komponent-id för armaturer i property set
- Omgång 2
    - Belysningsstolpar med 1D-geometri (IfcCircle)
    - Unika komponent-id för både stolpar och armaturer i property set

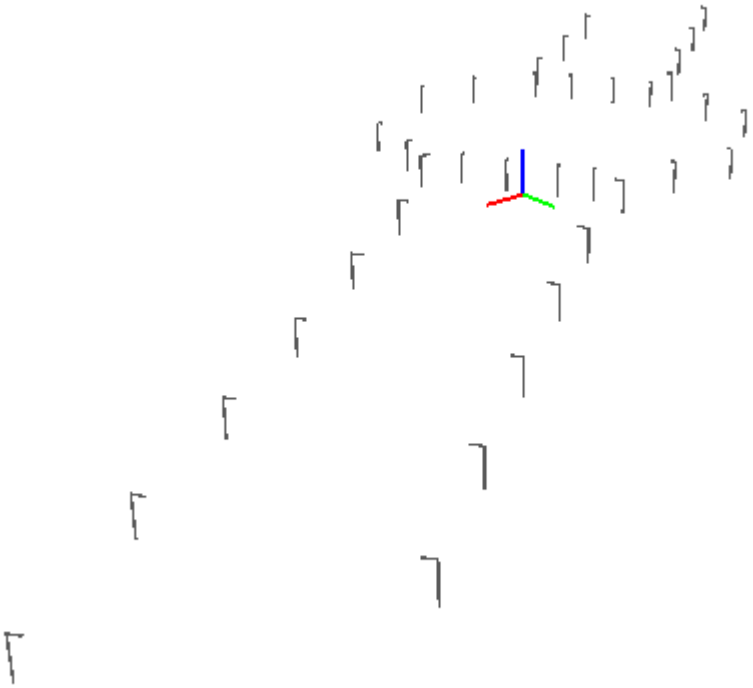
Separat levererades laddmall med egenskaper på excel-format. I excel-filen fanns komponent-id som gav möjlighet att koppla samman IFC-data med excel-data.

Processen för hantering av data gick till på följande sätt:

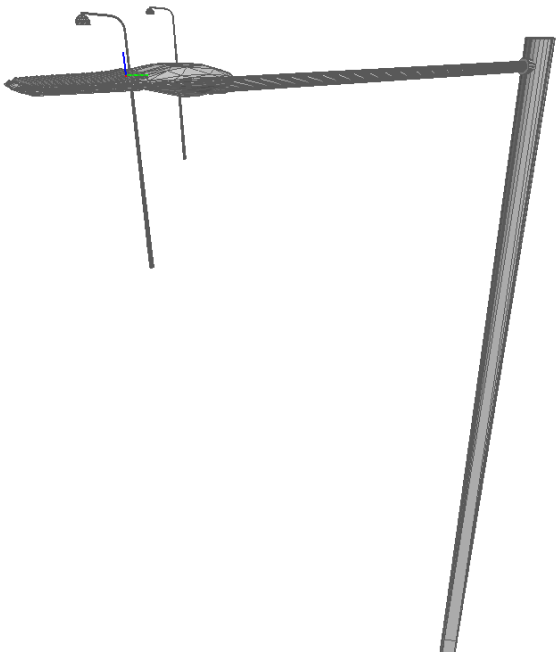
0. Import av berörda ontologier till default-graf
    a. IfcOWL
    b. OWL-ontologi motsvarande laddmall (se 0)
        i. Ska kanske egentligen modelleras enligt CoClass
    c. DCAT (för metadata)
1. Konvertering av IFC till RDF (Turtle) med hjälp av IFCtoRDF
2. Import av RDF till GraphDB till separat graf
3. Generering av metadata (semi-automatiskt)
4. Import av laddmall med hjälp av OntoRefine (verktyg i GraphDB för ”triplifiering” av tabell-data). Exempel kan ses nedan där kolumner i excel matchas mot ontologi (TRV_Laddmall).



5. Import av genererade RDF-data från excel till separat graf
6. Länkning av importerade excel-data med IFC-data över gemensamma komponent-id
    a. Med SPARQL skapas en trippel per objekt som länk mellan objekt och egenskapsdata
    b. Extra tripplar läggs i samma graf som egenskapsdata
    c. SPARQL för matchning listas i kapitel 0.
7. Skapa metadata
8. Demonstration av genomförd länkning med SPARQL



Figur 11 - Belysningsstolpar översikt



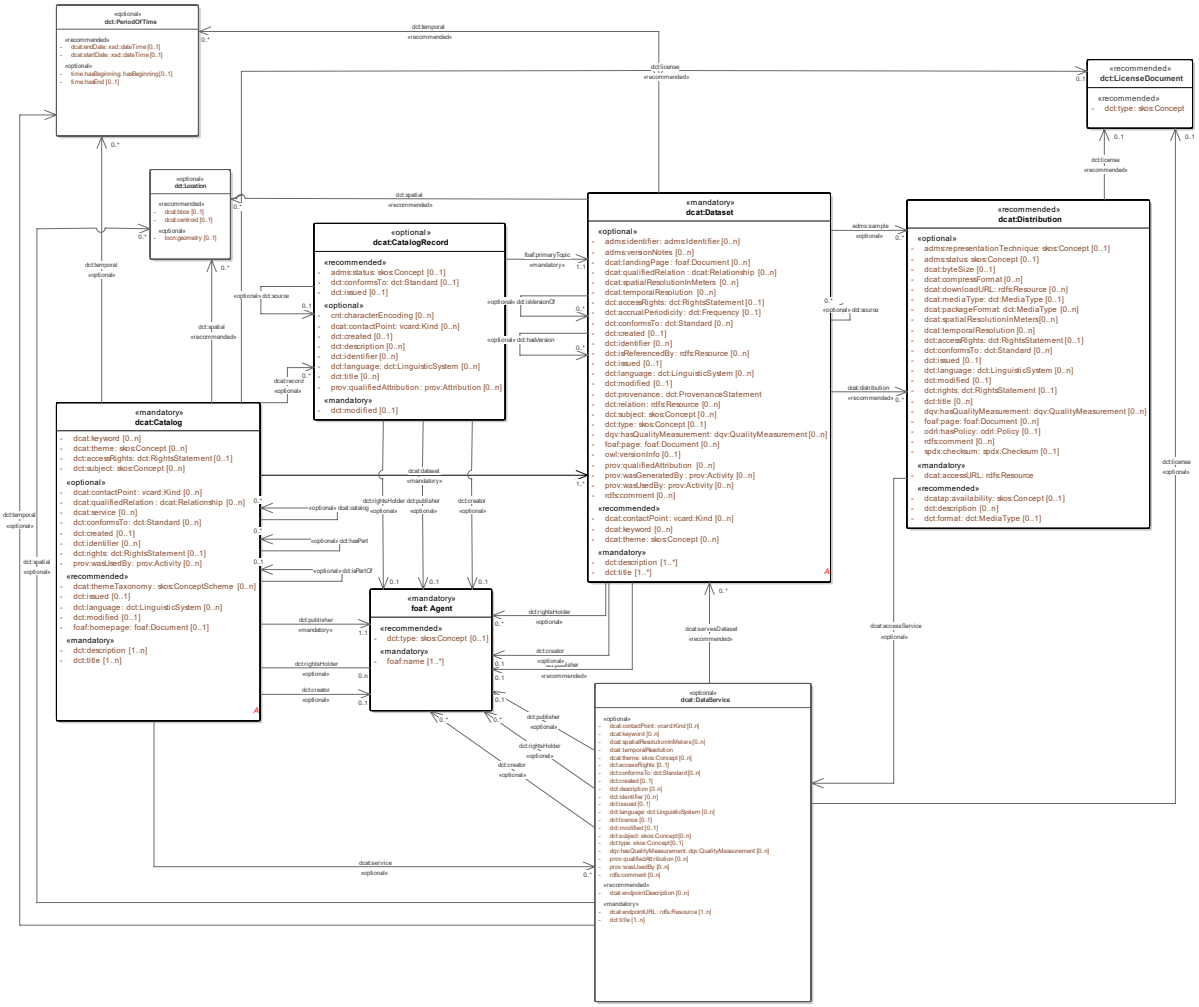
Figur 12 - Belysningsstolpe detalj

4.3 Vägräcken, vägmärken – Shape

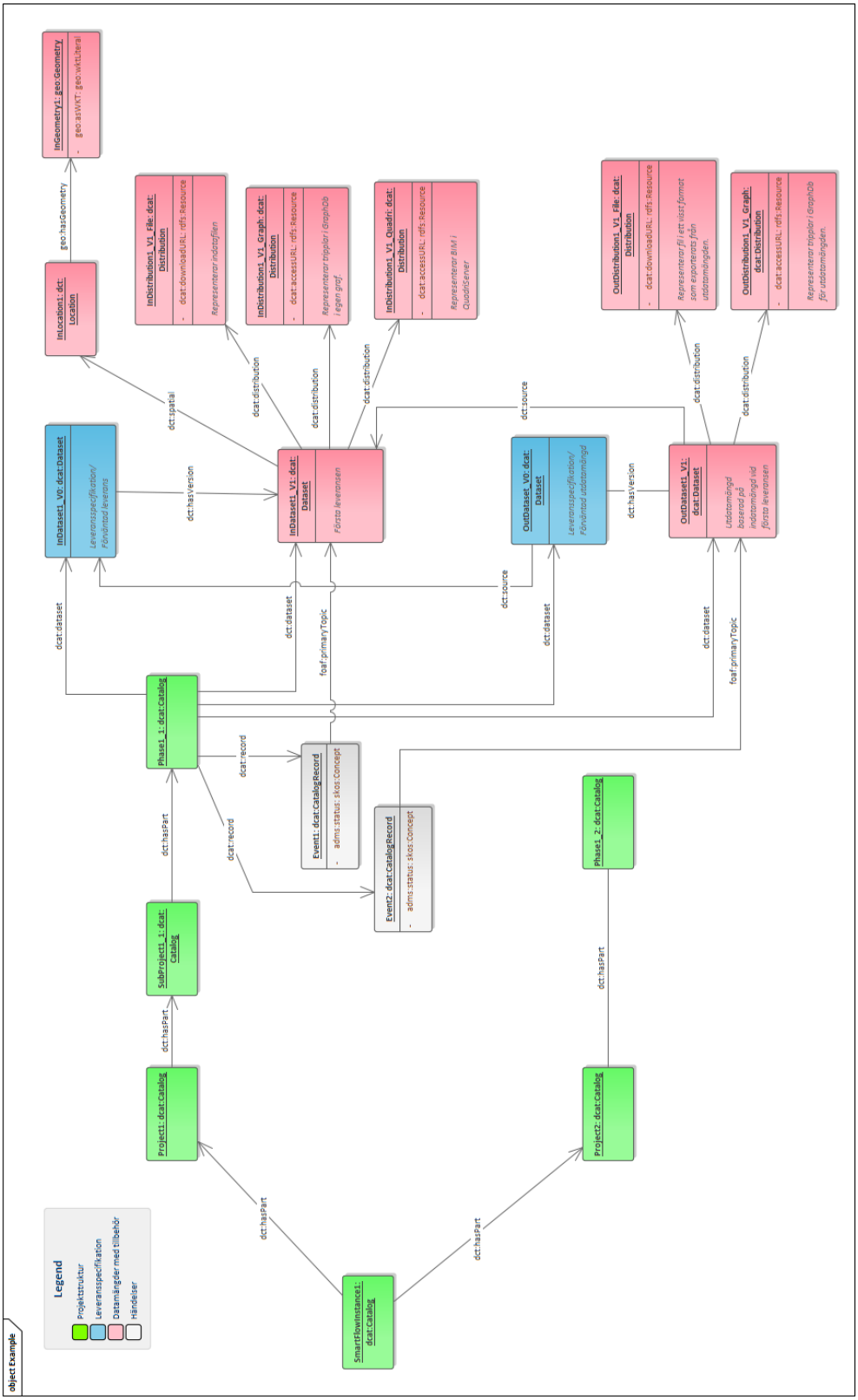
1. Import av Vägräcken, vägmärken etc från Shape/openTNF
    a. Med hjälp av TNE-tjänsten GenerateFeatures skapas data i OpenTNF (enligt definitioner i en datakatalog) med företeelser från Shape-fil.
2. Konvertering av OpenTNF katalogstruktur till OWL-ontologi (RDF).
    a. Med hjälp av OpenTNF-Library så läser vi in datakatalogen och de olika företeelserna med deras tillhörande attribut.
    b. Den fördefinierade ontologin för TNE importeras och mappas ihop med den inlästa datakatalogen.
    c. Tripplar skapas, med hjälp av dotNetRdf, som representerar datakatalogen och dess struktur.
    d. Den inlästa katalogen, företeelserna och företeelsernas attribut som nu är triplar skrivs med hjälp av dotNetRdf till RDF och sparas i en turtle-fil som kan importeras till GraphDB.
3. Instanser av företeelser i datakatalogen konverteras från OpenTNF till RDF
    a. Instanserna läses in med OpenTNF-Library och ontologin som representerar datakatalogens struktur importeras.
    b. Mappning sker mellan instanser i datakatalogen och ontologin med hjälp av OID och dotNetRdf.
    c. Tripplar skapas utifrån mappning till den importerade ontologin och attribut som instanserna har.
    d. Geometrin för den enskilda instansansen konverteras till geoSPARQL-geometri.
    e. Vägnätsknytning hanteras genom att varje länksekvens i vägnätet får sin egna unika URI som länkas till företeelsernas utbredning.
    f. De inlästa instanserna och geometri skrivs med hjälp a dotNetRdf till RDF och sparas i en turtle-fil som kan importeras till GraphDB.
4. De konverterade vägräckerna och vägmärkena (RDF) laddas in i GraphDB.
    a. Den sparade turtle-filen med instanserna laddas in i GraphDB.
    b. Metadata skapas

5 Appendix 2 - Metadata

5.1 GeoDCAT-AP - Version 2.0.0



5.2 Metadatastruktur, exempel



6 Appendix 3 – Använda ontologier

6.1 IfcOWL

IfcOWL (https://technical.buildingsmart.org/standards/ifc/ifc-formats/ifcowl/) tillhandahålls av BuildingSMART och är en exakt representation av IFC-schemat för att möjliggöra tvåvägs konvertering mellan STEP-format och RDF. Så här skriver BuildingSMART om IfcOWL:

”Using the ifcOWL ontology, one can represent building data using state of the art web technologies (semantic web and linked data technologies). IFC data thus becomes available in directed labelled graphs (RDF). This graph model and the underlying web technology stack allows building data to be easily linked to material data, GIS data, product manufacturer data, sensor data, classification schemas, social data, and so forth. The result is a web of linked building data that brings major opportunities for data management and exchange in the construction industry and beyond”

6.2 DCAT

DCAT (https://www.w3.org/TR/vocab-dcat-2/) är en rekommendation från W3C. Så här beskrivs DCAT:

”DCAT is an RDF vocabulary designed to facilitate interoperability between data catalogs published on the Web. This document defines the schema and provides examples for its use.

DCAT enables a publisher to describe datasets and data services in a catalog using a standard model and vocabulary that facilitates the consumption and aggregation of metadata from multiple catalogs. This can increase the discoverability of datasets and data services. It also makes it possible to have a decentralized approach to publishing data catalogs and makes federated search for datasets across catalogs in multiple sites possible using the same query mechanism and structure. Aggregated DCAT metadata can serve as a manifest file as part of the digital preservation process.”

SmartFlow använder DCAT som bas för att lagra metadata.

6.3 nvdb-owl

nvdb-owl (https://github.com/vegvesen/NVDB-Datakatalogen/tree/master/OWL) är en owl-representation av datakatalogen för NVDB i Norge. SmartFlow använder denna i samband med datakonverteringar vars utdata ska följa NVDB:s datakatalog.

6.4 TRV_Laddmall

I testfallet för belysning från Trafikverket erhölls en IFC-fil och separat en excel-fil med egenskaper. Excel-filen kunde importeras till GraphDB med hjälp av verktyget OntoRefine (https://graphdb.ontotext.com/documentation/free/loading-data-using-ontorefine.html). För att ”tag:a” excel-data skapades en ontologi som vi kallade ”TRV_Laddmall”. Ontologin listas här nedan. I ett verkligt case (kanske i arbetspaket 3) skulle man ha använt exempelvis egenskaper ur CoClass.

```
# baseURI: http://www.triona.se/SmartFlow/TRV_Laddmall
# prefix: TRV_Laddmall

@prefix TRV_Laddmall: <http://www.triona.se/SmartFlow/TRV_Laddmall#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://www.triona.se/SmartFlow/TRV_Laddmall>
  rdf:type owl:Ontology ;
  owl:versionInfo "Created with TopBraid Composer" ;
.
TRV_Laddmall:Amne1
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:Amne2
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:Armaturfabrikat
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:Armaturtyp
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:Beskrivning
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:CASnr1
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:CASnr2
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:Effektforbrukning
```

```
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:Entreprenad
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:Entreprenor
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:Fargtemperatur
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:FarligaAmnen
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:Fas
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:Felklass
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:Fundament
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:Fundamentdata
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:Halt1ViktProcent
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:Halt2ViktProcent
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:Installationsdatum
  rdf:type owl:DatatypeProperty ;
  rdfs:range xsd:integer ;
.
TRV_Laddmall:Kommun
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:KomplexAnlaggning
  rdf:type owl:DatatypeProperty ;
.
```

```
TRV_Laddmall:KomponentId
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:Lampa
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:Lampfabrikat
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:Lamptyp
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:Leverantor
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:Ljusflode
  rdf:type owl:DatatypeProperty ;
  rdfs:range xsd:float ;
.
TRV_Laddmall:MontageArArmatur
  rdf:type owl:DatatypeProperty ;
  rdfs:range xsd:integer ;
.
TRV_Laddmall:MontageArStolpe
  rdf:type owl:DatatypeProperty ;
  rdfs:range xsd:integer ;
.
TRV_Laddmall:Objekt
  rdf:type owl:Class ;
  rdfs:subClassOf owl:Thing ;
.
TRV_Laddmall:Objektnummer
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:Optiklage
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:OverordnadElkraft
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:PlaceringVagomrade
  rdf:type owl:DatatypeProperty ;
```

```
.
TRV_Laddmall:Startpunkt_X
  rdf:type owl:DatatypeProperty ;
  rdfs:range xsd:float ;
.
TRV_Laddmall:Startpunkt_Y
  rdf:type owl:DatatypeProperty ;
  rdfs:range xsd:float ;
.
TRV_Laddmall:Startpunkt_Z
  rdf:type owl:DatatypeProperty ;
  rdfs:range xsd:float ;
.
TRV_Laddmall:Status
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:Stolphojd
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:Stolptyp
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:Tillverkare
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:Vagnummer
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:Verifikation
  rdf:type owl:DatatypeProperty ;
.
TRV_Laddmall:harLaddmall
  rdf:type owl:ObjectProperty ;
  rdfs:range TRV_Laddmall:Objekt ;
.
```

7 Appendix 4 – SPARQL-exempel

7.1 Konvertering av Stikkrenner (IfcOWL) till nvdb-owl

```
PREFIX ifc: <https://standards.buildingsmart.org/IFC/DEV/IFC2x3/TC1/OWL#>
PREFIX list: <https://w3id.org/list#>
PREFIX expr: <https://w3id.org/express#>
PREFIX express: <https://w3id.org/express#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX nvdb-owl: <http://rdf.vegdata.no/nvdb/nvdb-owl#>
CONSTRUCT
{
    ?uri a nvdb-owl:vot79 .
    ?uri ?type ?object .
}
WHERE
{
    ?uri a ifc:IfcFlowSegment .
    ?pSetRel ifc:relatedObjects_IfcRelDefines ?uri .
    ?pSetRel ifc:relatingPropertyDefinition_IfcRelDefinesByProperties ?pSet .
    ?pSet ifc:hasProperties_IfcPropertySet ?pSetValue .
    ?pSetValue ifc:name_IfcProperty/expr:hasString ?name .
    ?pSetValue ifc:nominalValue_IfcPropertySingleValue ?value .
    OPTIONAL {
        ?value a ifc:IfcText .
        ?value expr:hasString ?valueStr .
        BIND(replace(?valueStr,"\\\\X2\\\\00E6\\\\X0\\\\","æ") as ?valueStr1)
        BIND(replace(?valueStr1,"\\\\X2\\\\00C5\\\\X0\\\\","Å") as ?valueStr2)
        BIND(replace(?valueStr2,"\\\\X2\\\\00F8\\\\X0\\\\","ø") as ?valueStr3)
        BIND(replace(?valueStr3,"\\\\X2\\\\00E5\\\\X0\\\\","å") as ?valueStrConv)
        FILTER (?name = "Cross-SectionShape")
        BIND(STRLANG(?valueStrConv,"no") as ?valueStrConvLang)
        BIND(nvdb-owl:et6984 AS ?type)
        ?object a nvdb-owl:kl6984 .
        ?object rdfs:label ?valueStrConvLang .
    }
    OPTIONAL {
        ?value a ifc:IfcText .
        ?value expr:hasString ?valueStr .
        BIND(replace(?valueStr,"\\\\X2\\\\00E6\\\\X0\\\\","æ") as ?valueStr1)
        BIND(replace(?valueStr1,"\\\\X2\\\\00C5\\\\X0\\\\","Å") as ?valueStr2)
```

```
        BIND(replace(?valueStr2,"\\\\X2\\\\00F8\\\\X0\\\\","ø") as ?valueStr3)
        BIND(replace(?valueStr3,"\\\\X2\\\\00E5\\\\X0\\\\","å") as ?valueStrConv)
        FILTER (?name = "InletType")
        BIND(STRLANG(?valueStrConv,"no") as ?valueStrConvLang)
        BIND(nvdb-owl:et1939 AS ?type)
        ?object a nvdb-owl:kl1939 .
        ?object rdfs:label ?valueStrConvLang .
    }
    OPTIONAL {
        ?value a ifc:IfcText .
        ?value expr:hasString ?valueStr .
        FILTER (?name = "InnerDiameter")
        BIND(xsd:integer(?valueStr) AS ?object)
        BIND(nvdb-owl:et3113 AS ?type)
    }
    OPTIONAL {
        ?value a ifc:IfcText .
        ?value expr:hasString ?valueStr .
        BIND(replace(?valueStr,"\\\\X2\\\\00E6\\\\X0\\\\","æ") as ?valueStr1)
        BIND(replace(?valueStr1,"\\\\X2\\\\00C5\\\\X0\\\\","Å") as ?valueStr2)
        BIND(replace(?valueStr2,"\\\\X2\\\\00F8\\\\X0\\\\","ø") as ?valueStr3)
        BIND(replace(?valueStr3,"\\\\X2\\\\00E5\\\\X0\\\\","å") as ?valueStrConv)
        FILTER (?name = "MainUsage")
        BIND(STRLANG(?valueStrConv,"no") as ?valueStrConvLang)
        BIND(nvdb-owl:et6981 AS ?type)
        ?object a nvdb-owl:kl6981 .
        ?object rdfs:label ?valueStrConvLang .
    }
    OPTIONAL {
        ?value a ifc:IfcText .
        ?value expr:hasString ?valueStr .
        BIND(replace(?valueStr,"\\\\X2\\\\00E6\\\\X0\\\\","æ") as ?valueStr1)
        BIND(replace(?valueStr1,"\\\\X2\\\\00C5\\\\X0\\\\","Å") as ?valueStr2)
        BIND(replace(?valueStr2,"\\\\X2\\\\00F8\\\\X0\\\\","ø") as ?valueStr3)
        BIND(replace(?valueStr3,"\\\\X2\\\\00E5\\\\X0\\\\","å") as ?valueStrConv)
        FILTER (?name = "MaterialType")
        BIND(STRLANG(?valueStrConv,"no") as ?valueStrConvLang)
        BIND(nvdb-owl:et6983 AS ?type)
        ?object a nvdb-owl:kl6983 .
        ?object rdfs:label ?valueStrConvLang .
    }
    OPTIONAL {
```

```
        ?value a ifc:IfcText .
        ?value expr:hasString ?valueStr .
        FILTER (?name = "OperationYear")
        BIND(xsd:integer(?valueStr) AS ?object)
        BIND(nvdb-owl:et4556 AS ?type)
    }
}
order by str(?uri) ?name
```

7.2 Selektion av ”NVDB-vy” för Stikkrenner_Hellefossen

```
PREFIX ifc: <https://standards.buildingsmart.org/IFC/DEV/IFC2x3/TC1/OWL#>
PREFIX list: <https://w3id.org/list#>
PREFIX expr: <https://w3id.org/express#>
PREFIX express: <https://w3id.org/express#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX nvdb-owl: <http://rdf.vegdata.no/nvdb/nvdb-owl#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
SELECT ?uri ?tvarsektion ?inlopp ?diam ?anv ?mtrl ?ar ?geom
WHERE
{
   ?uri a nvdb-owl:vot79.
   ?uri geo:hasGeometry/geo:asWKT ?geom .
   ?uri nvdb-owl:et6984/rdfs:label ?tvarsektion .
   ?uri nvdb-owl:et1939/rdfs:label ?inlopp .
   ?uri nvdb-owl:et3113 ?diam .
   ?uri nvdb-owl:et6981/rdfs:label ?anv .
   ?uri nvdb-owl:et6983/rdfs:label ?mtrl .
   ?uri nvdb-owl:et4556 ?ar .
}
```

7.3 Länkning av egenskaper från laddmall till IFC-objekt

```
PREFIX base: <http://example.com/base/>
PREFIX TRV_Laddmall: <http://www.triona.se/SmartFlow/TRV_Laddmall#>
PREFIX ifc: <http://standards.buildingsmart.org/IFC/DEV/IFC4_1/OWL#>
PREFIX expr: <https://w3id.org/express#>
PREFIX inst: <http://linkedbuildingdata.net/ifc/resources20210426_141628/>
INSERT
{
   GRAPH <http://www.smartflow.se/Belysning_IFC4x1_NP_2020_Props_v12>
   {
        ?ifcUri TRV_Laddmall:harLaddmall ?uriProp .
    }
}
WHERE
{
   GRAPH <http://www.smartflow.se/Belysning_IFC4x1_NP_2020_Props_v12>
   {
      ?uriProp TRV_Laddmall:Beskrivning ?beskr ;
                TRV_Laddmall:KomponentId ?kompId .
      BIND(CONCAT(SUBSTR(?kompId,3,6),SUBSTR(?kompId,10)) as ?kompIdX)
      FILTER(?kompIdX = ?value)
      {
         SELECT ?ifcUri ?name ?value
         WHERE
         {
            graph <http://www.smartflow.se/Belysning_IFC4x1_NP_2020_v12>
            {
               ?ifcUri a ifc:IfcLightFixture .
               ?pSetRel ifc:relatedObjects_IfcRelDefinesByProperties ?ifcUri .
               ?pSetRel ifc:relatingPropertyDefinition_IfcRelDefinesByProperties ?pSet .
               ?pSet ifc:name_IfcRoot/expr:hasString ?pSetName .
               ?pSet ifc:hasProperties_IfcPropertySet ?pSetValue .
               ?pSetValue ifc:name_IfcProperty/expr:hasString ?name .
               ?pSetValue ifc:nominalValue_IfcPropertySingleValue/expr:hasString ?value .
               FILTER(?name = "Komponent-id")
            }
         }
      }
   }
}
```

7.4 Generera bounding-box för metadata från IFC

```
PREFIX ifc: <https://standards.buildingsmart.org/IFC/DEV/IFC2x3/TC1/OWL#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX list: <https://w3id.org/list#>
PREFIX expr: <https://w3id.org/express#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX dcat: <http://www.w3.org/ns/dcat#>
PREFIX dct: <http://purl.org/dc/terms/>

CONSTRUCT {
    ?location geo:hasGeometry [
      rdf:type geo:Polygon ;
      geo:asWKT ?wkt ;
      geo:dimension 2 ;
      geo:coordinateDimension 2
    ]
}
WHERE {
  SELECT ?location (STRDT(CONCAT("<http://www.opengis.net/def/crs/EPSG/0/",?crsCode,"> POLYGON((",
STR(?minCoordE)," ",STR(?minCoordN), ",", STR(?minCoordE)," ",STR(?maxCoordN), ",",STR(?maxCoordE)," 
",STR(?maxCoordN), ",",STR(?maxCoordE)," ",STR(?minCoordN), ",",STR(?minCoordE)," ",STR(?minCoordN), "))"),
geo:wktLiteral) as ?wkt) ?crsCode
  WHERE {
    SELECT ?location (MIN(?coordE) as ?minCoordE) (MIN(?coordN) as ?minCoordN) (MAX(?coordE) as ?maxCoordE)
(MAX(?coordN) as ?maxCoordN) (SAMPLE(?crsCode) as ?crsCode)
    WHERE {
      SELECT ?location ?point ?coordE ?coordN ?crsCode
      WHERE {
        GRAPH <http://www.smartflow.se/Stikkrenner_Hellefossen/metadata> {
          ?location a dct:Location .
        }
        GRAPH <http://www.smartflow.se/Stikkrenner_Hellefossen> {
          OPTIONAL {
            ?proj a ifc:IfcProject .
            ?proj ifc:representationContexts_IfcContext ?ctx .
            ?mapConv ifc:sourceCRS_IfcCoordinateOperation ?ctx .
            ?mapConv ifc:eastings_IfcMapConversion/expr:hasDouble ?origoE .
            ?mapConv ifc:northings_IfcMapConversion/expr:hasDouble ?origoN .
            ?mapConv ifc:targetCRS_IfcCoordinateOperation/ifc:name_IfcCoordinateReferenceSystem/expr:hasString
?epsg .
          }

          ?point ifc:coordinates_IfcCartesianPoint ?coordListE .
```

```
            ?coordListE list:hasContents/expr:hasDouble ?relCoordE .
            ?coordListE list:hasNext ?coordListN .
            ?coordListN list:hasContents/expr:hasDouble ?relCoordN .

            BIND(COALESCE(?origoE + ?relCoordE,?relCoordE) as ?coordE)
            BIND(COALESCE(?origoN + ?relCoordN,?relCoordN) as ?coordN)
            BIND(COALESCE(STRAFTER(?epsg, ":"),"3006") as ?crsCode)

            FILTER(?coordE > 0 && ?coordN > 0)
          }
        }
      }
      GROUP BY ?location
    }
}
```