Slutrapport

Projekt SmartFlow

Dokumenthistorik:

Version	Initialer	Datum	Kommentar
1.0	LWi, KWe, AHu, HBl	2022-02-10	Slutrapport efter genomförd granskning

Innehållsförteckning

Sammanfattning

Detta dokument utgör slutrapport för det Vinnova finansierade projektet **SmartFlow**. Dokumentet beskriver det arbete som utförts samt de lärdomar som dragits. Vidare beskrivs tankar och rekommendationer kring framtida arbete inom området där datainsamling till förvaltningssystem för infrastruktur kommer att behöva anpassas till ny teknik och nya standarder både när det gäller nyinvestering och inventering.

Projektets mål har framför allt varit att titta på dataflödet och den hantering som behöver ske mellan projekt för nyinvestering/inventering och de system för anläggningsförvaltning som finns hos infrastrukturägare. En viktig utgångspunkt har varit att infrastrukturbranschen är komplex där dataleveranser kan ske med hjälp av olika standarder för olika discipliner och där förvaltningssystem ofta är specialiserade inom olika områden vilket leder till att de datastrukturer som används ofta är specialiserade. För att begränsa omfattningen i projektet har vi valt att specifikt titta på indataleveranser i form av IFC för nyinvestering, ibland kompletterat med data i excel-format, och shape för inventering.

Teknologierna och standarderna kring länkade data och semantisk web beskrivs som en möjlig väg framåt för att åstadkomma interoperabilitet och dataintegration. Standarderna är framtagna inom ramen för W3C och har som syfte att på ett enhetligt sätt överföra och beskriva data så att både människor och maskiner ska kunna tolka dessa. En förutsättning för lyckad kommunikation mellan parter, vare sig det gäller mänsklig eller maskinell kommunikation, är överenskommen och enhetlig syntax och semantik och båda dessa delar erbjuds av dessa standarder. Mot bakgrund av detta har en viktig del i SmartFlow-projektet varit att testa huruvida dessa teknologier kan erbjuda ett värdefullt bidrag till de dataflöden som undersöks i projektet.

Sammanfattningsvis kan sägas att, utifrån de tester som utförts, ansatsen har fungerat bra men att det fortfarande finns områden som behöver en djupare analys.

- Teknologin fungerar bra för automatiska översättningar mellan olika vokabulärer och strukturer/informationsmodeller, dvs för översättningar mellan olika informationsstrukturer, till exempel mellan IFC och den interna strukturen i ett förvaltningssystem.
- Teknologin medger att begreppssamband i hög grad kan beskrivas där domänkunskapen finns, dvs i verksamheten, i stället för att “gömmas” i olika informationssilo-system. Detta gör att dessa samband görs synliga för verksamheten vilket bör underlätta för denna att ta ägarskap över sina data
- För att fungera krävs ingående kunskaper kring de standarder som används för dataleveranser och de informationsstrukturer som används i en specifik verksamhet. Detta är dock ingenting nytt under solen. Skillnaden är sättet som dessa kunskaper görs tillgängliga och kan användas.
- Stora datamängder i kombination med omfattande och komplexa datastrukturer (scheman) innebär påfrestningar för prestanda i denna typ av lösning. IFC är ett exempel på en omfattande och komplex datastruktur. I detta fall kan man överväga att använda fokuserade delar av IFC-schemat (MVD:er) som grund i stället för det kompletta schemat.
- En annan påfrestning för denna typ av lösning är den många gånger komplexa och omfattande geometribeskrivning som förekommer när det rör sig om 3D-representation av komplexa objekt. Vi ser det inte som meningsfullt i dagens läge att använda länkade data och semantisk web för att bearbeta denna typ av geometri. Detta gör att följande slutsatser kan dras:
    - Fullständig 3D-geometri bör tas omhand separat i miljöer som klarar att lagra, visualisera och på andra sätt bearbeta och analysera denna typ av geometri.

    - Fullständig geometri bör lagras på sådant sätt att den enkelt kan länkas till de objekt som bär geometrin. OGC GeoSPARQL är en standard på GIS-området där man tagit fram lösningar för hur man kan länka samman geometriska beskrivningar med de objekt som använder dessa beskrivningar

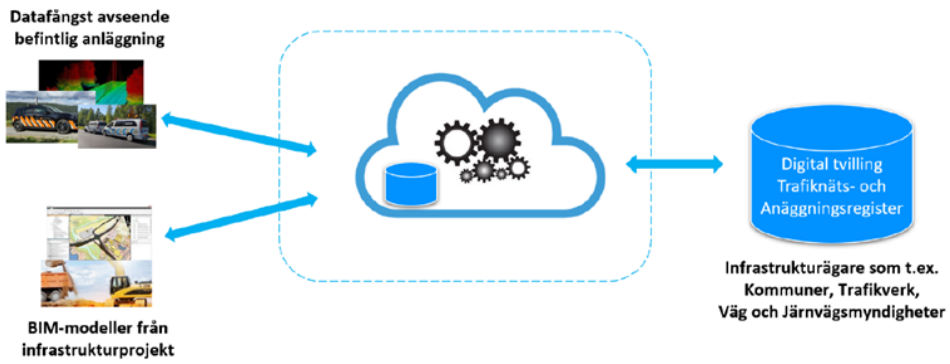
- För hantering av de flöden som normalt förekommer mellan nyinvesteringsprojekt och förvaltning finns det normalt ett behov att förenkla geometrin så att avancerad 3D-geometri generaliseras som punkter, linjer och ytor för att ge information om det geografiska läget för en tillgång. Denna typ av enkel geometrisk representation är den som normalt används vid inventering, t ex via fotografering eller laserscanning. Metoder för geometriförenkling beskrivs separat i detta dokument, men är ett område i utveckling.
- Inom långsträckt infrastruktur finns oftast ett behov att beskriva tillgångarnas läge relativt ett nätverk för att enkelt kunna samköras med annan information, t ex trafikmängder, trafikrestriktioner och –regler, trafikolyckor etc. Därför behöver den generaliserade geometrin ofta även översättas till ett läge relativt nätverket (väg- eller järnvägsnätet).
    - Hur dessa lägen kan representeras beskrivs bland annat i SS637004 eller Nasjonal Vegdatabank.
    - För järnväg finns RailSystemModel 1.2 (tidigare RailTopoModel 1.1) som hanterar detta specifikt för järnvägsnät.
    - I IFC-standarden finns inte nätverksrepresentationen explicit. Däremot finns möjlighet att specificera linjära lägen längs väg- och spårlinjer (IfcAlignment). För att kunna utnyttja detta i samband med trafiknätsbaserade objektlägen så krävs att det finns en tydlig mappning mellan en väg- eller spårlinje och motsvarande länk i trafiknätet.
- Sökbara metadata utgör en central komponent för ett system som SmartFlow. Att beskriva förväntade och faktiska leveranser i form av krav och andra egenskaper utgör en grund för ett kvalitetssäkrat arbetssätt. I projektet har vi använt ontologin GeoDCAT-AP för detta men det finns även andra ontologier som kan vara relevanta.
    - Som metadata ingår även de specifikationer (maskinläsbara) som ligger till grund för maskinell verifiering av levererade data. Exempel på maskinläsbara specifikationer kan vara EXPRESS-scheman (som för IFC), MVD:er, XML-scheman eller OWL-ontologier tillsammans med SHACL-regler.
- Det är i skrivande stund oklart vad som utgör den bästa ansatsen för verifiering av IFC- och andra typer leveranser.
    - MVDXml finns att tillgå, men hur kan dessa tas fram på bästa sätt? Hur är tillgången på programvara som kan utföra maskinell verifiering baserat på MVDXml? MVDXml är också anpassat specifikt för IFC. För data som inte ryms i IFC-världen gäller det att hitta andra sätt för maskinell verifiering.
    - En möjlig ansats kan vara att bygga verifiering baserat på SHACL. Det kan även vara en möjlighet att generera SHACL utifrån MVDXml. SHACL har fördelen att man då utför kontroller mot RDF-data vilket möjliggör verifiering av data som har sitt ursprung utanför IFC. Detta behöver dock testas mera

1 Inledning

1.1 Syfte och målgrupp

Detta dokument är en del av arbetspaket 4 i projektet SmartFlow. Tillsammans med demonstrationer och workshops utgör dokumentet den slutrapportering som görs från projektet.

Projektet har haft som mål att utveckla, testa och demonstrera processer och en teknisk lösning för att möjliggöra automatiserade, effektiva och öppna informationsflöden rörande data om transportinfrastruktur. Dessa informationsflöden rör framförallt förekommande flöden mellan projekt för nyinvestering/ny infrastruktur och inventering av befintlig infrastruktur och de förvaltnings-/tillgångssystem som finns hos infrastrukturägare.



Figur 1 - SmartFlow – översikt

En central teknologi som använts för ansatsen är **länkade data** och **semantisk web**. Teknologierna runt länkade data och semantisk web har som ansats att lägga grunden för dataintegration och förenkling av informationsutbyte mellan system.

Tidigare rapporter har levererats från arbetspaket 1 och 2. Därför fokuserar denna rapport på det arbete som utförts i arbetspaket 3 samt resultat och slutsatser därifrån. Vidare diskuteras vad resultat och slutsatser kan innebära för framtida arbete på området och eventuell produktifiering.

I arbetspaket 3 har vi framförallt fokuserat på att implementera en testmiljö för att kunna genomföra beslutade testfall för att därigenom dra slutsatser kring relevansen för om och hur den tekniska ansatsen är relevant för de informationsflöden som testats.

Dokumentet är primärt ägnat åt projektets parter samt styrgrupp samt övriga som har ett intresse av denna typ av lösning.

1.2 Referenser

1. Rapport arbetspaket 1, SmartFlow
2. Rapport arbetspaket 2, SmartFlow
3. https://semiceu.github.io/GeoDCAT-AP/releases/2.0.0/
4. https://www.w3.org/2011/rdf-wg/wiki/Main_Page
5. https://www.w3.org/TR/rdf-schema/
6. https://www.w3.org/TR/owl-guide/
7. https://www.w3.org/TR/shacl/
8. https://www.w3.org/TR/sparql11-query/
9. https://www.ogc.org/standards/geosparql
10. https://technical.buildingsmart.org/standards/ifc/
11. https://technical.buildingsmart.org/standards/ifc/ifc-formats/ifcowl/
12. https://coclass.byggtjanst.se/login
13. https://www.kartverket.no/geodataarbeid/standardisering/sosi-standarder2
14. https://www.opentnf.org/
15. https://github.com/vegvesen/NVDB-Datakatalogen/tree/master/OWL
16. https://www.ontotext.com/products/graphdb/
17. https://www.esri.com/content/dam/esrisites/sitecore-archive/Files/Pdfs/library/whitepapers/pdfs/shapefile.pdf
18. https://developer.tekla.com/trimble-connect/documentation/trimbim-technology

1.3 Definitioner och förkortningar

Term/förkortning	*Definition*
SmartFlow Server	*Konceptuellt system etablerat som begrepp i AP1. Ska utgöra ett inleveransstöd med syfte att kunna automatisera dataflöden mellan infrastrukturprojekt (både ny- och ombyggnation och inventering/inmätning) och anläggningsförvaltningen.* *Termen används i dokumentet både för det konceptuella systemet och för den implementerade demo-versionen av systemet som tagits fram av projektet.*
RDF	*Resource definition framework* *https://www.w3.org/TR/rdf11-concepts/*
RDFS	*Resource definition framework schema* *https://www.w3.org/TR/rdf-schema/*

OWL	*Web ontology language* *https://www.w3.org/TR/owl-ref/*
SHACL	*Shape's constraint language* *https://www.w3.org/TR/shacl/*
IFC	*Industry foundation classes* *https://www.iso.org/standard/70303.html* *https://www.buildingsmart.org/standards/bsi-standards/industry-foundation-classes/*
MVD	*Model View Definition* *Model View Definitions (MVD) är buildingSMART's lösning för att skapa IFC-baserade standarder som kan implementeras och testas.* *En MVD består av tre huvudkomponenter:* - *En mängd Concept Templates som definierar överenskommelser om hur IFC-schemat ska användas.* - *En mängd Exchange Requirements. Ett urval av entiteter och egenskaper som är relevanta för ett urval av användningsfall.* - *En beskrivning av hur mjukvara ska hantera de data som överförs.* *https://www.buildingsmart.org/standards/bsi-standards/model-view-definitions-mvd/* *MVD:er kan uttryckas enligt MVDXml som är ett xml-schema.* *https://technical.buildingsmart.org/standards/ifc/mvd/mvdxml/*
CoClass	*Klassifikationssystem för den byggda miljön.* *https://byggtjanst.se/tjanst/coclass*
Informationsmodell	*En strukturerad beskrivning av den information som behövs i en verksamhet. Informationsmodellen bör använda sig av de termer och begrepp som definierats i en begreppsmodell.*
Begreppsmodell	*En strukturerad beskrivning av de begrepp och tillhörande termer som används i en verksamhet och deras samband.*
NVDB	*Nationell vägdatabas (Trafikverket)/Nasjonal vegdatabank (Statens Vegvesen)* *https://www.trafikverket.se/tjanster/data-kartor-och-geodatatjanster/las-om-vara-data/vagdata/* *https://www.vegvesen.no/fag/teknologi/nasjonal-vegdatabank/*
GUS	*Gemensamt underhållsstöd (Trafikverket)*

	https://www.trafikverket.se/tjanster/system-och-verktyg/forvaltning-och-underhall/gus---gemensamt-underhallsstod/
BaTMan	*Bridge and tunnel management (Trafikverket)* *https://batman.trafikverket.se/externportal*
Ontologi	*En formell beskrivning av koncept och relationer inom en domän. Synonymt med TBOX.*
Kanonisk modell	*En kanonisk modell är en typ av datamodell som beskriver objekt, relationer och egenskaper i enklast möjliga form. En kanonisk modell används typiskt för att förenkla integrationer av system och databaser där all dataöverföring sker över den kanoniska modellen som därför utgör en gemensam beskrivning av data.* *Anm: För svenska testfall i SmartFlow-projektet har vi använt oss av CoClass (omgjord till OWL-representation) som kanonisk modell.*
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]*
Triplar	*Den atomära dataentiteten i RDF. Som namnet indikerar så är en triple en mängd av tre entiteter som kodar ett faktauttryck i formen subjekt-predikat-objekt (t ex "Bob is 35", or "Bob knows John"). En mängd med triples bildar tillsammans en kunskapsgraf.*
Länkade data	*En metod för att publicera strukturerade 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 om-klassificera objekt och egenskaper utifrån att ontologier länkats samman via relationer med en definierad och standardiserad logik.*

2 Projektets genomförande

2.1 Förutsättningar

Följande parter har deltagit i projektet:

- Triona AB (koordinator)
- Trimble AB
- Ramboll AB
- Trafikverket (behovsägare)
- Statens vegvesen (behovsägare)

Utöver det så har Svensk Byggtjänst deltagit genom att tillhandahålla licenser för CoClass.

Arbetet i projektet har bedrivits i fyra arbetspaket:

- Arbetspaket 1
    - Tidsperiod: 1 september 2020 – 31 december 2020
    - Mål: Planering samt specifikation av lösning och dataflöden
- Arbetspaket 2
    - Tidsperiod: 1 januari 2021 – 30 april 2021
    - Mål: Genomföra ”Proof of Concept” (POC)
- Arbetspaket 3
    - Tidsperiod: 1 maj 2021 – 31 oktober 2021
    - Mål: Ta fram fördjupad lösning
- Arbetspaket 4
    - Tidsperiod: 1 november 2021 – 31 december 2021
    - Mål: Sammanställning och redovisning

Som input till arbetet i arbetspaket 3 hade vi rapporten från arbetspaket 2 [2]. Kortfattat så definierades där följande aktiviteter:

- Utveckla och beskriv arkitektur
    - Förfina use cases/user stories
    - Definiera egenskapskrav
    - Definiera principer för lagring av data och metadata
    - Definiera principer för datatransformationer
    - Specificera komponenter och GUI
    - Specificera deployment (driftsättningsmiljö)
    - Specificera kart- och 3D-tjänster
    - Definiera scope för demonstrationer

- Utveckla lösning för GUI för demonstrationer
    - Utveckla lösning för geometriförenkling
    - Utveckla mappning till topologiskt nät
    - Utveckla hantering av koordinatsystem/transformationer
- Ta fram representativa testfall
- Specificera användning av CoClass
    - Definition av kanonisk modell

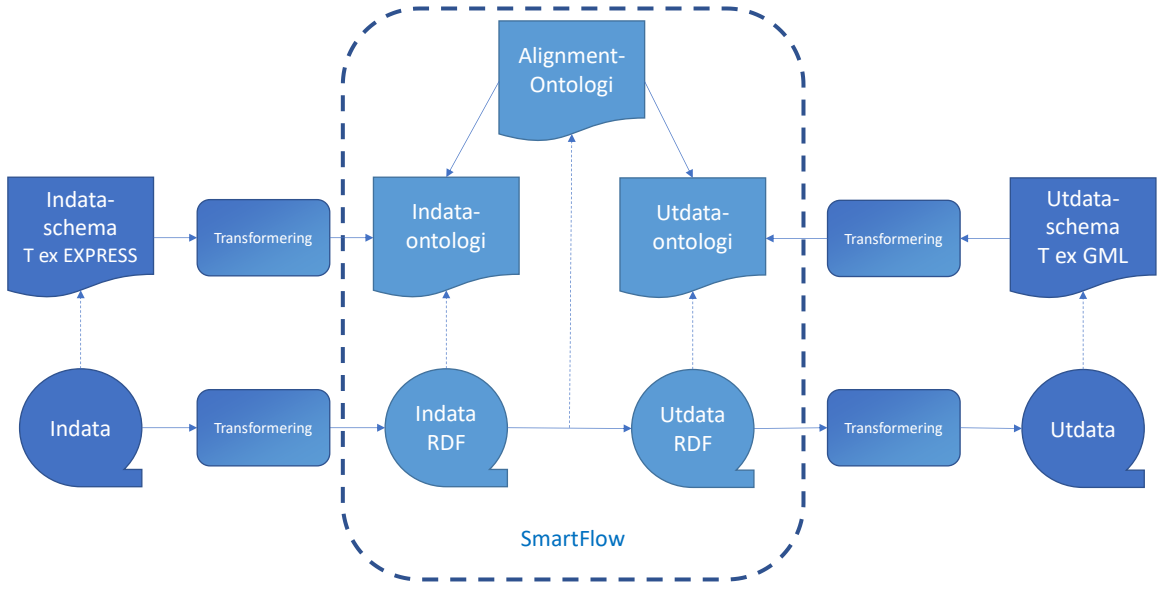
2.2 Aktiviteter

2.2.1 Informationsmodellering

2.2.1.1 Principer för datatransformationer

Eftersom SmartFlow använder sig av standarder för länkade data och semantisk web för att beskriva och överföra information så behöver de datastrukturer (scheman) som används för in- och utdata beskrivas i form av ontologier med hjälp av OWL och RDFS. På samma sätt behöver data från olika format konverteras till RDF (en mängd tripplar). När data och ontologier som beskriver data på en gemensam form enligt RDF, RDFS och OWL finns, så kan data med hjälp av alignment-ontologier (se separat kapitel nedan) transformeras mellan olika in- och utdatastrukturer.

Ett exempel för detta för IFC redovisas i figuren nedan.



Figur 2 - Principer översikt

2.2.1.2 Ontologier för in- och utdata

Som beskrivs ovan behöver scheman för in- och utdata i SmartFlow beskrivas i form av ontologier enligt OWL och RDFS. I några fall finns sådana ontologier redan tillgängliga och i andra fall har funktionalitet utvecklats för att automatiskt generera ontologier från andra typer av scheman enligt nedanstående tabell:

Schema/standard	Status	Kommentar
IFC2.3, IFC4	Officiell ontologi finns	https://technical.buildingsmart.org/standards/ifc/ifc-formats/ifcowl/
Datakatalogen norsk NVDB	Officiell ontologi finns	https://github.com/vegvesen/NVDB-Datakatalogen/tree/master/OWL
CoClass	Officiell ontologi saknas	Automatisk konverteringsrutin från CoClass api till ontologi har utvecklats (se separat kapitel nedan)
Trafikverket-ontologi	Officiell ontologi saknas	Separat ontologi som kompletterar/detaljerar CoClass har byggts inom ramen för projektet
Ontologi för GUS	Officiell ontologi saknas	Ontologi har byggts manuellt utifrån excel-format som erhållits från GUS-projektet.
Ontologi för NVDB/STVDB	Officiell ontologi saknas	Automatisk konverteringsrutin från datakatalog i OpenTNF-format till ontologi har utvecklats inom ramen för projektet.
Ontologier för indata i shape-format (inventeringsdata)	Officiell ontologi saknas	Datakataloger som motsvarar indata har beskrivits i OpenTNF-format varvid den automatiska rutinen som beskrivs på föregående rad har kunnat användas.

2.2.1.3 ”Kanonisk” ontologi baserad på CoClass

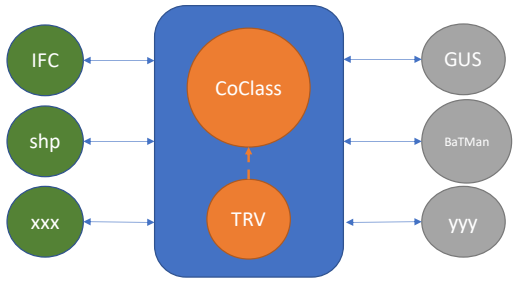
Inom ramen för SmartFlow så har vi haft tillgång till CoClass och CoClass api. En applikation har tagits fram som anropar CoClass api och genererar en ontologi enligt följande principer:

1 Varje Klass i CoClass blir en owl:Class
2 Subklasser i CoClass definieras som subklasser i owl med rdfs:subClassOf
3 Varje Egenskap i CoClass blir en owl:DatatypeProperty (eller owl:ObjectProperty)
4 Sub-egenskaper i CoClass definieras som sub-properties i owl med rdfs:subPropertyOf
5 Vi definierar en URI för varje klass och egenskap samt knyter till övriga meta-egenskaper (t ex namn, definition, CoClass-kod) till respektive definition.

Eftersom CoClass har separata taxonomier för klasser och egenskaper och i sig inte har obligatoriska restriktioner för vilka egenskaper som ska knytas till varje klass så lämpar sig en owl-ontologi över CoClass utmärkt som grundplåt för en kanonisk modell att använda vid översättningar mellan olika datascheman.

På indatasidan finns exempelvis IFC (i olika versioner – t ex IFC 2.3, IFC 4 och IFC 4.3) samt andra datastrukturer från exempelvis inventering. På utdatasidan finns olika förvaltningssystem med sina unika datastrukturer. Genom att använda CoClass så får vi en situation där varje datastruktur beskriver sina samband med CoClass i stället för att alla olika datastrukturer ska beskriva sina samband med alla andra.

I vissa fall går CoClass-klassificeringen inte tillräckligt djupt, dvs innehåller inte tillräckligt specifika begrepp för att direkta begreppssamband kan beskrivas. För att kunna hantera dessa fall så har en separat ontologi för Trafikverket tagits fram. Trafikverkets ontologi länkar mot och bygger ut CoClass och lägger till Trafikverks-specifika begrepp för att på så sätt bygga ut CoClass taxonomi till en tillräcklig detaljeringsnivå (tillräckligt specifika begrepp).

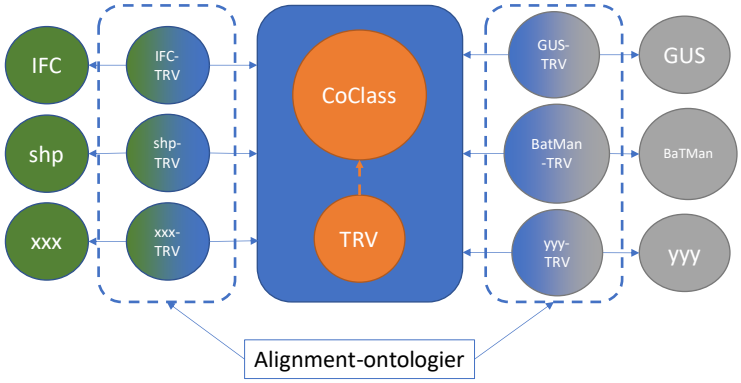


Figur 3 - Användning av kanonisk ontologi (TRV/CoClass)

2.2.1.4 Alignment-ontologier

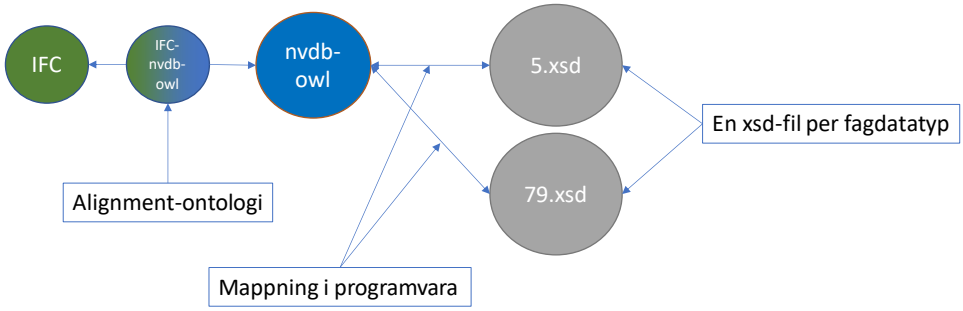
Alignment-ontologier är separata ontologier som länkar begrepp från andra ontologier. Om vi t ex vill beskriva hur begrepp i IFC 4.3 förhåller sig till begrepp i CoClass så görs detta bäst genom att i en separat ontologi beskriva dessa länkar. Detta kan enkelt göras utan att de ontologier som länkas på något sätt behöver påverkas. Genom att begrepp från olika ontologier länkas med hjälp av standardiserade sambandstyper från RDFS och OWL så kan sedan programvara automatiskt översätta data med hjälp av så kallad reasoning.

Bilden från föregående kapitel kan därför kompletteras med alignment-ontologier för att ge en mera fullständig bild.



Figur 4 - Användning av kanonisk ontologi (TRV/CoClass) med alignment-ontologier

För testfall för Statens Vegvesen är situationen enklare eftersom indata är IFC och utdata ska vara enligt NVDB:s datakatalog. Dock ska data levereras enligt ett separat GML-schema. Detta har vi hanterat på följande sätt:



Figur 5 - Alignment-ontologi för Statens Vegvesen

XML-scheman för NVDB:s datakatalog har hämtats från https://github.com/vegvesen/NVDB-Datakatalogen/tree/master/GML. "Mappning i programvara" har för enkelhets skull gjorts via egenutvecklad kod med C#. För denna typ av mappning torde det bästa på sikt vara att hitta en generell programvara som klarar automatisk översättning mellan OWL och XSD respektive RDF och XML.

2.2.2 Testmiljö och utvecklingsmiljö

Kapitlet beskriver de programvaror och komponenter som använts i och utgör utvecklingsmiljö och testmiljö för SmartFlow-projektet. I de fall komponenterna ingår i demo-versionen av SmartFlow Server (exempelvis Ontotext GraphDB) så beskrivs det hur i kapitel 2.2.4.

2.2.2.1 Microsoft Teams

För samarbete inom projektet med samtliga parter har Microsoft Teams (Team: SmartFlow) använts.

2.2.2.2 Microsoft DevOps

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

2.2.2.3 Microsoft Azure

I Microsoft Azure har vi etablerat en 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 och AP3. Där har vi också installerat övriga komponenter (egenutvecklade och tredje-parts) som behövts för att kunna demonstrera SmartFlow som ett resultat av AP3.

Vi har installerat två instanser av den utvecklade demo-versionen av SmartFlow Server; en instans för att testa och demonstrera med testdata från Sverige och en instans för att testa och demonstrera med testdata från Norge.

Tack vare molnarkitekturen (Infrastructure as a Service, IaaS) är det lätt att skala upp/ut utvecklingsmiljön vid behov inom ramen för de ekonomiska förutsättningarna. Lösningen att använda IaaS i molnet innebär också att vi har tillgång till en en dator-miljö utan att behöva införskaffa hårdvara för projektet med allt vad det kan innebära i form av kostnader, väntetid och administrativ insats.

Vi har inte använt några andra PaaS-tjänster i Azure (Platform as a Service vilket innebär att vi har visat att ett SmartFlow-system, med de komponenter vi har utvecklat och använt i AP3 kan köras ”on-premise”, hos en driftleverantör med egen datorhall, med fysiska eller virtuella datorer om alternativet med molndrift inte är aktuellt.

2.2.2.4 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 projektets testmiljö. Projektet har via avtal även haft tillgång till 3 dagars development support.

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

Ett par motiv för val av GraphDB är:

1. GraphDB har använts av delar av projekt-teamet tidigare vilket har gjort det lättare att komma igång
2. 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

GraphDB har som sagt 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 (reasoner-programvaran) att automatiskt klassificera data genom att man på schema-nivå lägger till information om hur olika begrepp förhåller sig till varandra.

Som frågespråk har GraphDB stöd för SPARQL, inklusive GeoSPARQL som medger 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. Detta medför att man behöver tänka igenom sin repository-struktur i de fall att man vill separera känsliga data med begränsad åtkomst från andra data. Eftersom RDF hanteras som tripplar så kan data separeras på egenskapsnivå. Det kan t ex vara fördelaktigt om ett och samma subjekt har vissa egenskaper som är känsliga och behöver separeras ut.

2.2.2.5 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 2.2.3.

2.2.2.6 Apache Kafka och RabbitMQ

För att hantera kommunikationen mellan olika komponenter i SmartFlow Server har vi använt en meddelande-baserad arkitektur och provat två olika meddelandeplattformar.

Projektet började med en Confluent-distribution av Apache Kafka (https://www.confluent.io/product/confluent-platform/) med motivet att kunna testa ett antal intressanta funktioner, t.ex. den i GraphDB inbyggda möjligheten till integrationen med Kafka samt möjligheten att nyttja Kafka som databas. Kafka är också distribuerad, skalbar och högpresterande.

Givet att projektet inte hann testa ovan nämnda funktioner samt kompatibilitetsutmaningar mellan Apache Kafka och testmiljön valde vi att använda RabbitMQ som meddelandeplattform i den slutliga demo-versionen av SmartFlow Server. Även RabbitMQ går att köra distribuerat över flera maskiner vid behov för horisontell skalning och felsäkerhet och implementerar de grundfunktioner för meddelande-kommunikation mellan komponenter som vi efterfrågat.

Confluent-distributionen av Apache Kafka är vad Confluent kallar för Open Core, fri att använda men inte fri att vidareutveckla (https://www.confluent.io/confluent-community-license-faq/). RabbitMQ är Open Source (https://www.rabbitmq.com/mpl.html).

2.2.2.7 PostGIS/PostgreSQL

För att enkelt kunna visa data, som levererats till SmartFlow Server, i en karta har vi använt PostGIS/PostgreSQL för att lagra de geografiska egenskaperna hos levererat data.

PostGIS/PostgreSQL lagrar och tillgängliggör de geografiska egenskaperna enligt OGC Simple Feature for SQL-standarden.

PostGIS/PostgreSQL är att betrakta som Open Source (https://postgis.net/workshops/postgis-intro/license.html) (https://www.postgresql.org/about/licence/)

2.2.2.8 Trimble Web3D

Web3D är ett grafikbibliotek, utvecklat av Trimble, med återanvändbara och utökningsbara komponenter för att visualisera och redigera 3D-modeller.

SmartFlow har nyttjat valda delar av biblioteket för att i SmartFlow Server kunna visualisera de 3D-modeller som levereras som indata till systemet.

Web3D har stöd för flera olika 3D-format men SmartFlow har valt att använda formatet TrimBim för att visualisera 3D-modeller i SmartFlow Server.

För mer information om TrimBim-formatet, se [18].

2.2.2.9 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 ramverk som Microsoft ser som huvudspåret för .NET-utveckling (vid tidpunkten när detta dokument skrivs har Microsoft lanserat .NET6). För front end-utveckling har vid behov också JavaScript använts.

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).

2.2.2.10 Utvecklingsmiljö SmartFlow Server

Utvecklingen av SmartFlow Server har skett i Microsoft Visual Studio 2019 på projektmedlemmarnas egna datorer och använda tredjepartskomponenter (beskrivna ovan) har i största möjliga mån körts i en lokal container-miljö (Docker for Windows) på respektive utvecklingsdator. GraphDB har dock bara installerats i testmiljön där den också använts under utveckling.

2.2.2.11 Verktyg för att skapa och redigera ontologier

Inom SmartFlow-projektet har vi, förutom att återanvända redan färdiga ontologier, skapat ontologier på några olika sätt:

- Manuellt i en dedikerad ontologi-editor. De editorer som vi använt är dels TopBraid Composer (https://www.topquadrant.com/products/topbraid-composer/) samt Protege (https://protege.stanford.edu/). TopBraid Composer kostar pengar medan Protege är open-source (BSD). Även Enterprise Architect, som normalt används som UML-verktyg kan användas för att modellera OWL-ontologier. Detta har inte använts i SmartFlow och anledningen till detta är främst avsaknad av erfarenhet med verktyget för just ontologi-

modellering. För att kunna beskriva en detaljerad ontologi enligt OWL-standarden så krävs anpassningar av UML för detta eftersom UML i grunden inte har alla konstruktioner som finns i OWL. För de som gärna använder Enterprise Architect rekommenderas noggranna tester som vi inte ansåg oss ha tid för inom ramen för SmartFlow-projektet.

- Genom att skriva konverteringsprogram i C#. Detta har använts för CoClass vars data finns tillgängliga via ett proprietärt API. I detta fall har en automatisk rutin för konvertering till OWL skrivits i C# genom användning av program-biblioteket DotNetRDF som är open-source (MIT). Biblioteket är ett komplett bibliotek för att hantera, läsa skriva mm RDF-data.

2.2.2.12 Verktyg för att skapa RDF-data

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

I SmartFlow Server anropas IFCtoRDF automatiskt från en inom projektet utvecklad komponent som ett led i en indata-leverans.

För Excel-data använder vi verktyget OntoRefine (https://graphdb.ontotext.com/documentation/free/loading-data-using-ontorefine.html) som medföljer GraphDB (se nedan). OntoRefine kan konvertera diverse format (t ex tabulära data som Excel) till RDF enligt en specifikation som man själv kan definiera och återanvända.

För övriga data använder vi DotNetRDF och egenskrivna C#-program för konvertering till och från RDF.

2.2.3 Testfall dataflöden

För att verifiera tekniken som används i SmartFlow server har ett antal testfall tagits fram. Dessa listas i nedanstående tabell. En mera detaljerad beskrivning finns i kapitel **Fel! Hittar inte referenskälla.**.

Testfall	Format	Innehåll	Resultat
Stikkrenner, Testdata Hellefossen **Statens Vegvesen**	IFC 4x1	Stikkrenner i form av IfcPipeSegment Egenskaper i PropertySet: - PredefinedType - InletType - MainUsage - MaterialType - OutletType - Cross-sectionType - InnerDiameter - OperationYear	XML enligt Sosi GML-schema för inläsning i NVDB Norge
Rekkverk, Testdata Hellefossen **Statens Vegvesen**	IFC 4x1	Rekkverk i form av IfcRailing Egenskaper i PropertySet: - PredefinedType - AdditionalInformation - BuiltYear - FenceType - Location	XML enligt Sosi GML-schema för inläsning i NVDB Norge

		- PoleDistance - Usage	
Belysningsstolpar + TRV Laddmall med egenskaper **Testdata BIM Trafikverket**	IFC 4x1 csv (laddmall med egenskaper)	Belysningsstolpar i form av IfcMember (stolpe) och IfcLightFixture (Armatur) Egenskaper i PropertySet: - Komponent-id Egenskaper i separat excel enligt TRV laddmall.	Excel för inläsning i GUS
Broräcke **Testdata BIM Trafikverket**	**IFC 4x1**	**Broräcke i form av IfcRailing** **Egenskaper i PropertySet:** - **PredefinedType** - **Kapacitetsklass** - **Leverantör** - **Modellbeteckning**	**Open TNF för inläsning i TNE enligt TRV specifikation**
Belysningsstolpar **Testdata Inventering Trafikverket**	Shape (konverteras till OpenTNF i TNE innan inläsning i SmartFlow Server)	Punkter	Excel för inläsning i GUS
Vägräcken **Testdata inventering Trafikverket**	Shape (konverteras till OpenTNF i TNE innan inläsning i SmartFlow Server)	Linjer	Open TNF enligt TRV specifikation för inläsning i TNE

2.2.4 Utveckla SmartFlow Server

2.2.4.1 Bakgrund och syfte

Redan i AP1 ([1]) etablerades begreppet SmartFlow Server som ett konceptuellt system för att utgöra ett inleveransstöd med syfte att kunna automatisera dataflöden mellan infrastrukturprojekt (både ny- och ombyggnation och inventering/inmätning) och anläggningsförvaltningen.

Under AP3 utvecklade SmartFlow-projektet en demo-version av SmartFlow Server med det primära syftet att (förutom det, genom namnet, uppenbara att på ett begripligt sätt kunna demonstrera SmartFlow för projektets intressenter)

- Testa principer och tekniker kring hypotesen att använda semantiska webben som en central del i det automatiserade dataflödet

Sekundärt var också förhoppningen att kunna

- Väcka tankar kring hur ett system som SmartFlow Server skulle kunna fungera
- Ställa frågor kring arkitekturen för SmartFlow Server

2.2.4.2 Arkitekturella krav

Det har, i projektet SmartFlow, aldrig varit en prioriterad aktivitet att samla in kvalitetskrav (icke-funktionella krav) för systemet SmartFlow Server. Dock skickades det ut en enkät till projektets behovsägare, Trafikverket och Statens Vegvesen, för att få ”mellan tummen och pekfinger”-uppgifter på datamängder och säkerhet.

Värt att notera baserat på input till och analys i projektet:

- Den momentana genomströmningshastigheten för olika aktiviteter i leveransprocessen är inte en begränsande faktor
- Datamängder som ska hanteras av systemet varierar kraftigt i storlek, givet
    - Filstorlek och komplexitet i data
    - Antal leveranser över en given tidsperiod
    - Livslängden hos data i systemet
- Syftet med systemet har aldrig varit att agera slutförvar; när data som levererats till systemet har konsumerats kan det tas bort från SmartFlow Server
- Datamängder som levereras till systemet kan vara föremål för sekretess
- Exekveringmiljön för systemet är inte given men det finns uttryckt ett behov av att kunna köra systemet ”on premise” på virtuella eller fysiska datorer

2.2.4.3 Användningsfall

Baserat på arbetet i AP1 finns ett antal tänkbara användningsfall för SmartFlow Server.

I demo-versionen har vi valt att fokusera på följande:

- Leveransprocessen (givet ett antal testfall, se kapitel 2.2.3)
    - En användare levererar en datamängd
    - Systemet registrerar datamängden
    - Systemet skapar en utleverans
- Användaren tittar på in- och utleveranser (metadata och data) i form av
    - Tabell
    - Karta
    - 3D-vy

Exempel på tänkbara användningsfall eller aktiviteter som inte har implementerats i demo-versionen av SmartFlow Server är

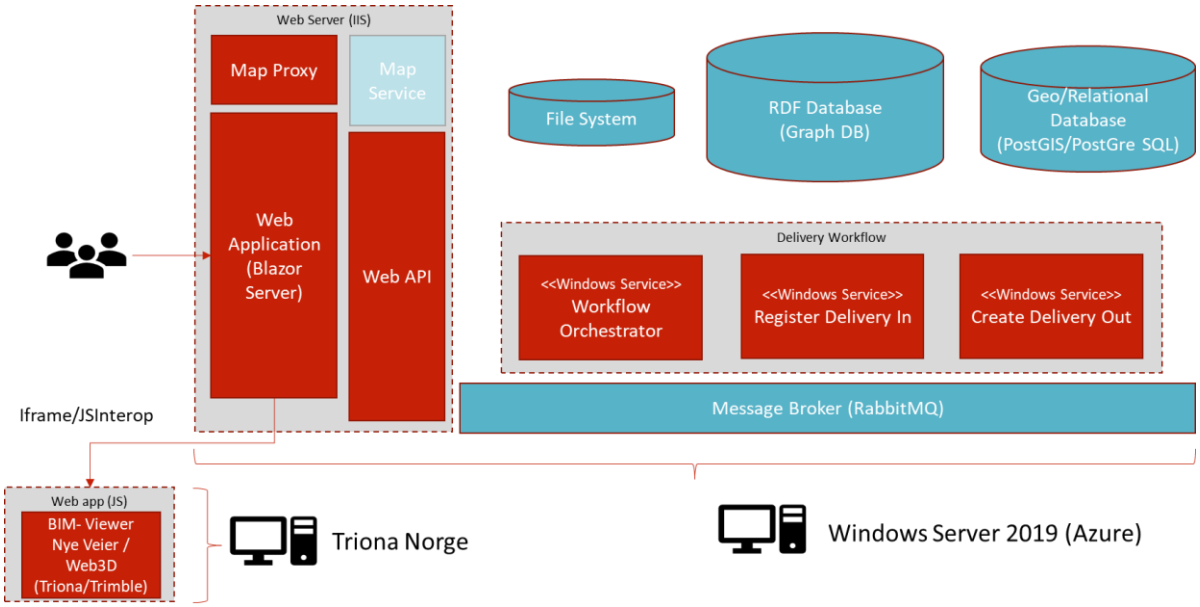
- Notifiering vid förändring av datamängd till olika intressenter
- Konsumera utleverans
- Beställa utleverans
- Administrera innehåll, exempelvis
    - Skapa nya projekt och delprojekt
    - Registrera leveransspecifikationer
- Lägga till nya transformationer

För demo-versionen av SmartFlow Server har administrationen av innehållet lösts genom möjligheten att manuellt exekvera ett antal fördefinierade SPARQL-skript, vilka skapar önskade projektstrukturer, registrerar leveransspecifikationer, lägger till kodlistor m.m.

2.2.4.4 Arkitektur

Givet syftet med demo-versionen av SmartFlow Server, fokus på en liten delmängd av tänkbara användningsfall och okända kvalitetskrav har vi valt en arkitektur som är

- Enkel att beskriva och förstå
- Flexibel



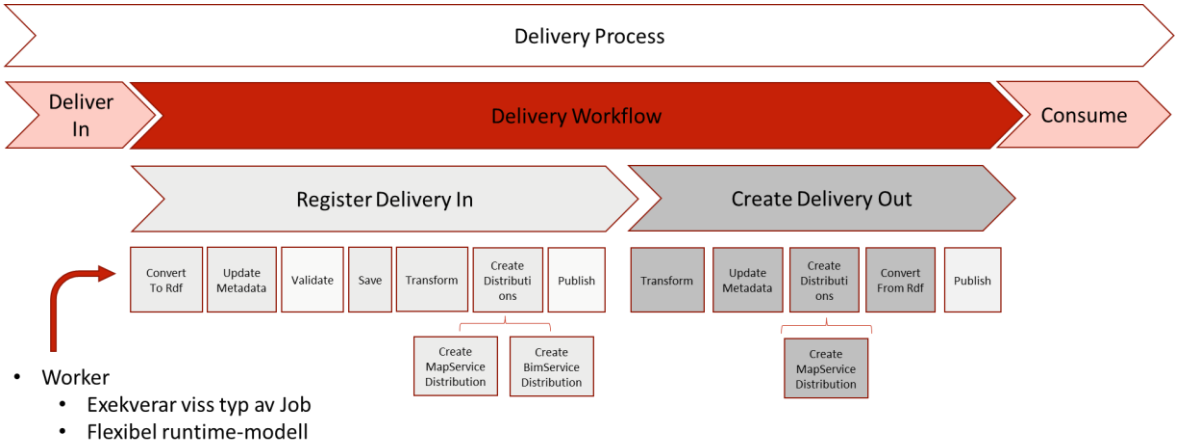
Figur 6 - Statisk modell över demo-versionen av SmartFlow Server

En statisk modell över demo-versionen av SmartFlow Server finns beskriven i Figur 6. De komponenter som utgör systemet är

- En webb-applikation vilken utgör användargränsnittet för att kunna leverera datamängder till systemet och titta på data och metadata i systemet
- En webb-applikation vilken utgör proxy för de karttjänster som används för olika kartvyer i användargränssnittet.
- Ett webb-api som exponerar funktioner och data som användargränssnittet behöver. Webb-api är komponenten som exponerar funktioner och data till externa system om sådana skulle finnas. Implementerar tillsammans med användargränssnittet inleverans-steget (Deliver In) i leveransprocessen
- Ett antal windows-tjänster eller på annat sätt startade windows-processer som implementerar arbetsflödet (Delivery Workflow) i leveransprocessen
- En meddelandeplattform som tar emot och levererar de meddelanden som driver leveransprocessen framåt
- En RDF-databas (GraphDB) med ansvar för att lagra levererade och skapade instans- och metadatamängder i RDF-format samt tillhandahålla funktioner för automatisk transformering och konvertering av tabelldata till RDF
- En Geo/Relationsdatabas med ansvar för att lagra instansdatamängder för att smidigt kunna visualisera (eller söka baserat på) de geografiska egenskaperna hos levererade och skapade instansdatamängder i en karta
- En webb-applikation för att kunna visualisera levererade instansdatamängder i 3D

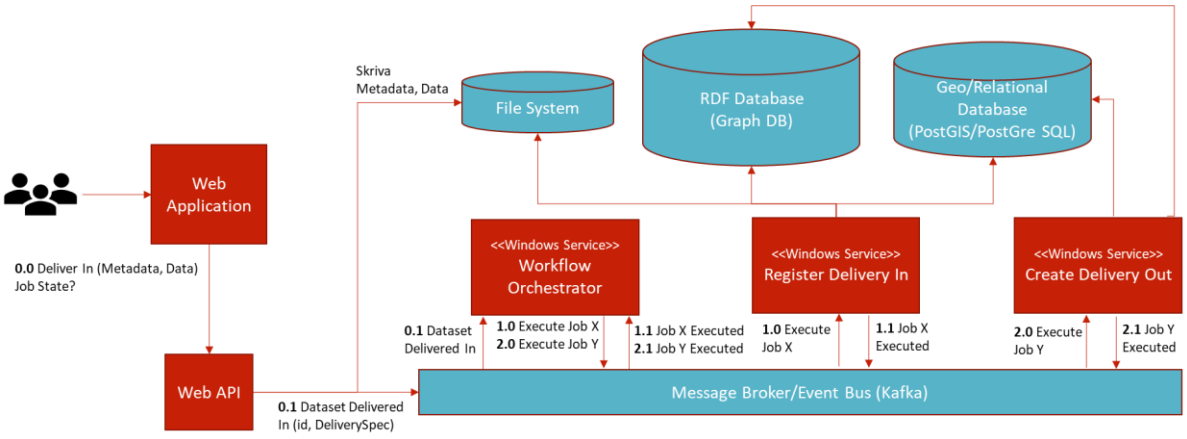
De röda komponenterna i Figur 6 är utvecklade inom SmartFlow-projektet, de blå komponenterna är konfigurerbara tredjeparts-komponenter eller tillgänglig infrastruktur i testmiljön.

Leveransprocessen så som den är definierad för och implementerad i demo-versionen av SmartFlow Server beskrivs i Figur 7. Observera att steget ”Consume” inte är implementerat (det prioriterades ned då det inte direkt bidrog till demo-systemets primära syfte, se kapitel 2.2.4.1).



Figur 7 - Leveransprocessen med sina aktiviteter så som den är definierad för demo-systemet

Figur 8 visar ett förenklat och abstrakt flödesdiagram över hur leveransprocessen har implementerats i demo-versionen av SmartFlow Server och hur olika komponenter interagerar med varandra. Numreringen av meddelanden i flödet (ex. 0.0, 0.1,...,2.1) visar i vilken sekvens olika komponenter används i flödet.



Figur 8 – Ett förenklat flödesdiagram för SmartFlow Server för leveransprocessen

Arkitekturen ger lösningar på saker som vi trots allt tror oss veta och behöva

- Varje aktivitet i leveransprocessen (se Figur 7) är implementerad som en egen modul vilka går att kombinera till olika exekvereningsenheter vilket ger enkelhet och flexibilitet. I demo-

systemet har vi valt att skapa exekveringsenheter av respektive delprocess RegisterDeliveryIn och CreateDeliveryOut
- Den implementerade leveransprocessen är meddelande-driven vilket ger möjlighet till löst kopplade moduler och en robust process som kan parallelliseras och distribueras
- Arkitekturen begränsar inte exekveringsmiljön. Demo-systemet kör alla ingående komponenter i en virtuell server i Azure vilket lika gärna skulle kunna vara en virtuell eller fysisk dator ”on-premise”. Det skulle också gå att nyttja andra former av exekveringsmiljöer (exempelvis Azure Functions) och nyttja PaaS-tjänster för olika komponenter i systemet
- Systemet kan partitionera data i vila för att om möjligt hantera olika skyddsvärden ur ett säkerhetsperspektiv för olika datamängder (se kapitel 2.2.6)
- Demo-systemet implementerar i liten skala begreppet ”Polyglot persistens” vilket innebär att man väljer olika lagringsmiljöer för olika typer av data och de behov som kan kopplas till olika typer av data. Nackdelen kan vara ökad komplexitet men fördelen är optimerad lagring för de behov som finns för olika typer av data. I demo-systemet innebär det lagring av geografiska data i en geo/relationsdatabas för visualisering med hög prestanda och standardiserade gränssnitt

Vi har också jobbat med en konfigurerbar design (med avseende på systemdesign, inte UI-design) för att underlätta användandet av systemet inom projektet men också som en tänkbar lösning på utmaningar som att exempelvis lägga till en ny specifikation (och transformation).

Projektet definierar konfigurering som en möjlighet att förändra systemet utan behov av att bygga en ny version av systemet. Att lägga till ett SPARQL-skript till systemet som exekverar vid ett specifikt tillfälle därför att skriptet följer en given namnsättning är därigenom en konfigurering.

Tillägg av en ny specifikation skulle då, förenklat, kunna innebära att man

- Lägger till ett nytt värde i kodlistan för specifikationer
- Lägger till en fil med ontologi för specifikationen
- Lägger till filer med alignment-ontologierna för de transformationer som behövs
- Lägger till filer med SPARQL-skript för de transformationer som behövs

Val av teknik och miljö för användargränssnittet är baserat huvudsakligen på två aspekter

- Webben är en rimlig och lättdistribuerad miljö för ett användargränssnitt
- Tillgänglig kompetens i projektet (användargränssnittet i demo-versionen av SmartFlow Server är en Blazor-applikation utvecklad i C#)

För att visa levererade datamängder i 3D används en webbapplikation i form av en paketering av valda delar av grafikbiblioteket Web3D utvecklad av Trimble.

Webbapplikationen laddas och visas i användargränssnittet för SmartFlow Server om datamängden som visas är en datamängd som kan visas i 3D. I demo-versionen innebär det att en (1) representation av datamängden måste vara en fil i formatet TrimBim och levereras med vid inleveransen av datamängden om sådan fil finns.

2.2.5 Metadata

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

2.2.5.1 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.

Projektet har också tittat på den svenska profilen av DCAT-AP, DCAT-AP-SE (https://docs.dataportal.se/dcat/).

Som påpekas i dokumentationen för DCAT-AP-SE så fokuserar dokumentation som finns kring GeoDCAT-AP på att beskriva en kompatibel metadataprofil som ska omfatta de delar som återfinns i ISO19139. Det är en ytterst liten delmängd av utökningen i GeoDCAT-AP i förhållande till DCAT-AP som har berörts i SmartFlow. Ett exempel är beskrivning av det spatiala referenssystemet för en geografisk datamängd.

DCAT verkar fylla de behov som SmartFlow Server har, framför allt vad gäller struktur. Utökningar har tillförts huvudsakligen av systemtekniska skäl.

Den stora vinsten i att använda DCAT är inte det interna bruket av metadata i SmartFlow Server utan möjligheten att exponera innehållet i SmartFlow Server i en, för sammanhanget, rimlig och känd specifikation som gör det möjligt för maskinell tolkning.

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

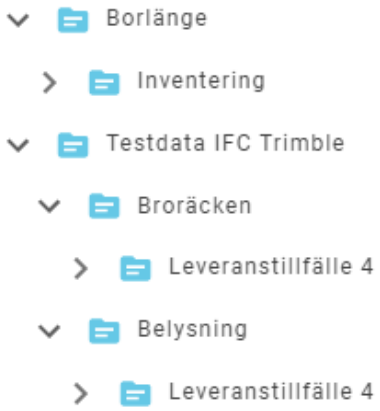
Se kapitel 6.1 för en modell över GeoDCAT-AP version 2.

Se kapitel 6.2 för ett modell-exempel på hur vi har använt GeoDCAT för att representera projektstruktur, leveransspecifikationer, datamängder och händelser.

2.2.5.2 Kataloger

För att ordna datamängder, som levererats till SmartFlow Server (indatamängder) eller skapats av SmartFlow Server efter transformation av indatamängder (utdatamängder), i en projekt-struktur har projektet 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 leveranstillfällen”. En grundare hierarki skulle kunna innebära att ”projekt har delprojekt”.

I Figur 9 finns ett exempel på projekt-struktur (noder i en trädstruktur) som utgått från fyra testfall i AP3. Bilden visar två projekt, Borlänge och Testdata IFC Trimble. I projekt Borlänge finns katalogen Inventering som representerar ett delprojekt för leverans av data efter inventering av vägräcken och belysning i Borlänge. I projekt Testdata IFC Trimble finns kataloger för delprojekt för data uppdelat på olika företeelser, broräcken och belysning, och i respektive delprojekt finns katalogen Leveranstillfälle 4 till vilka ett fiktivt nyinvesteringsprojekt leverera data för broräcken respektive belysning. Alla noder i Figur 9 representeras av varsin dcat:Catalog-instans.



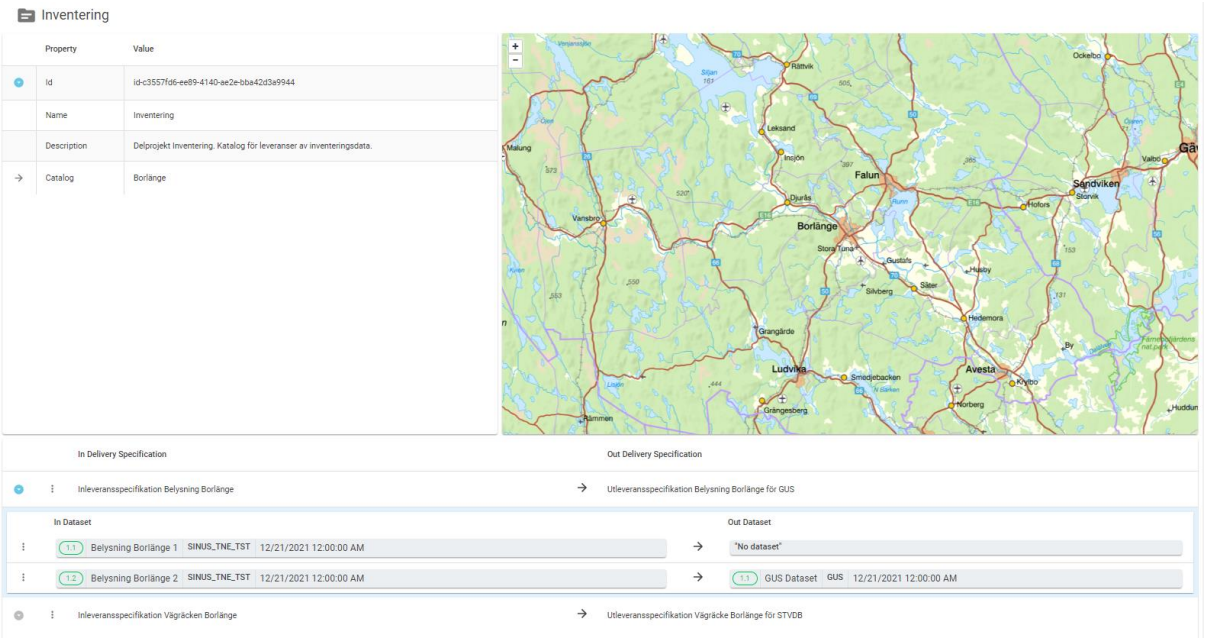
Figur 9 – Exempel katalogstruktur med projekt, delprojekt och leveranstillfällen

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 9 är det löven i en gren, 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 10 visas ett exempel med information och innehåll för Borlänge/Inventering.

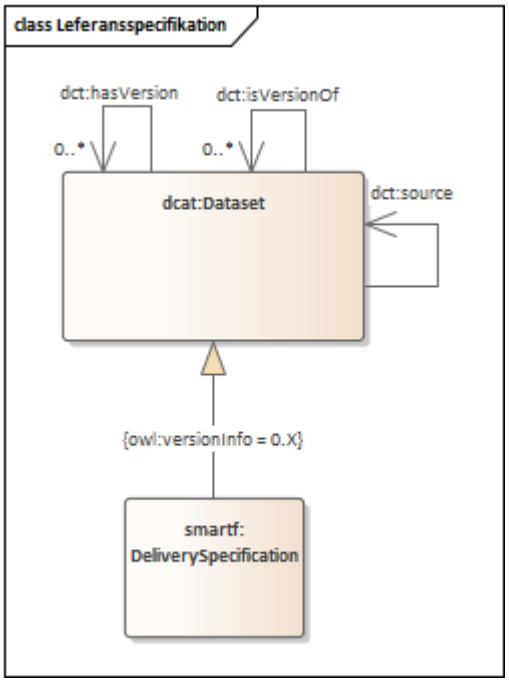


Figur 10 - Information om och innehåll i Inventering, Borlänge (skärmbild från SmartFlow Server)

2.2.5.3 Leveransspecifikationer

En indatamängd är tänkt att uppfylla en viss inleveransspecifikation och en utdatamängd skapas utifrån en viss utleveransspecifikation.

Vi har valt att representera en leveransspecifikation som en utökning av klassen dcat:Dataset. Leveransspecifikationen representerar dock inte en faktisk datamängd i SmartFlow utan beskriver en förväntad datamängd och vad som förväntas av den (ex. vilka filformat som ska levereras eller skapas, vem som ska leverera, när data ska levereras, geografisk utbredning, etc.).



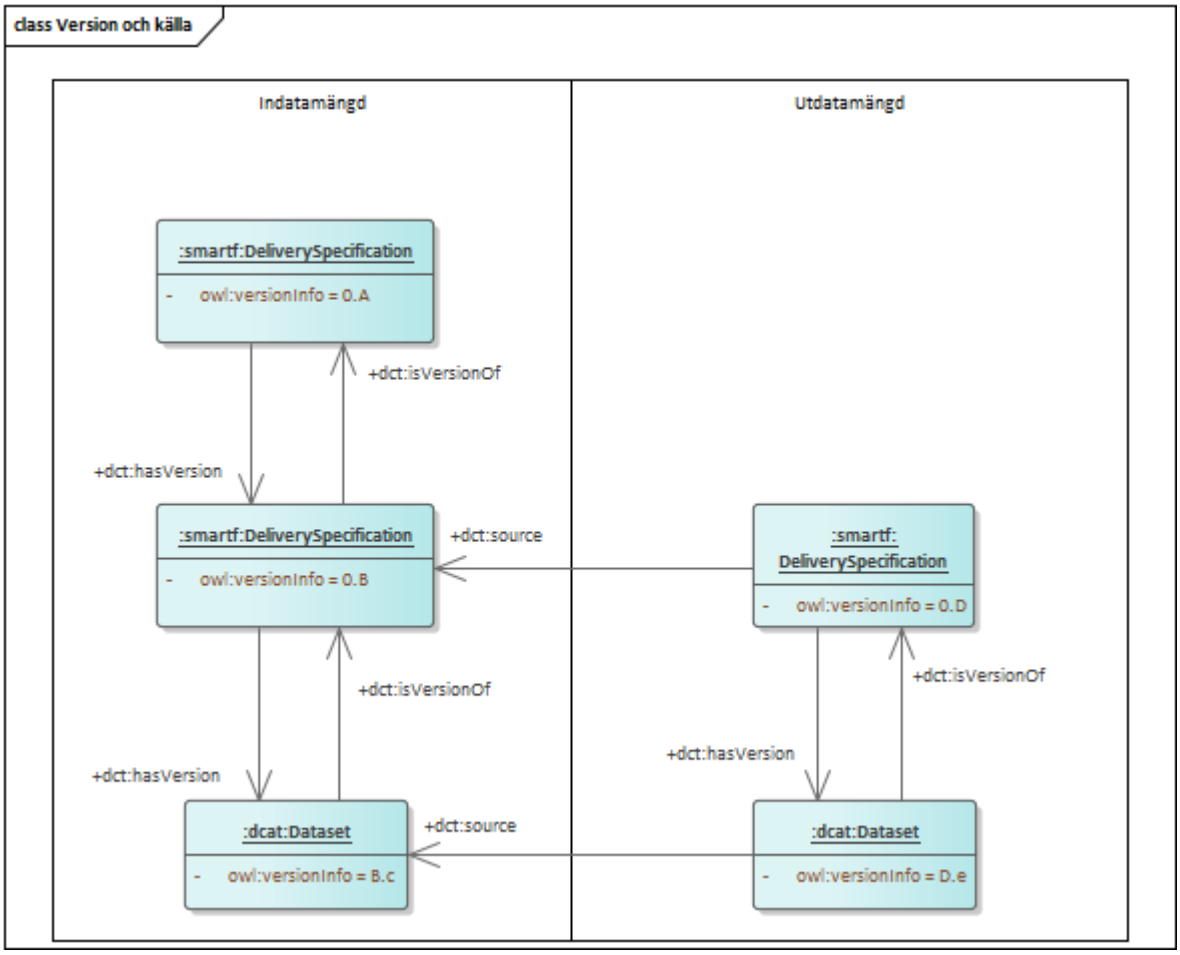
Figur 11 - Leveransspecifikation som en utökning av dcat:Dataset

I och med att en leveransspecifikation är ett dcat:Dataset så ser vi den också som en del av innehållet i en katalog. Dvs. i SmartFlow Server registrerar man en leveransspecifikation i den katalog där datamängder som motsvarar leveransspecifikationen också kommer att registreras. Ur ett DCAT-perspektiv blir då leveransspecifikationen version 0 av en viss datamängd.

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.

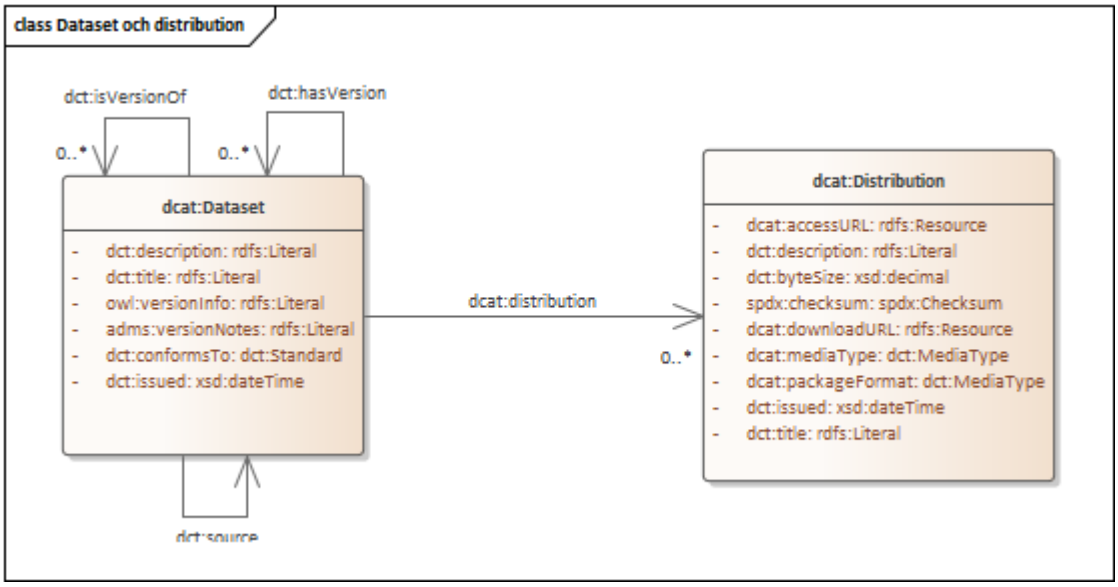
Då både leveransspecifikationer och datamängder representeras av instanser av dcat:Dataset kommer de att delvis ha samma egenskaper. Skillnaden ligger då i tolkningen. Ett exempel är publicist (objektet i en trippel av typen dcat:Dataset dct:publisher foaf:Agent, se modellen kapitel 6.1); för leveransspecifikationen 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. När informationen om leveransspecifikationer och datamängder presenteras i ett gränssnitt som SmartFlow Server kan gränssnittet hjälpa till med den tolkningen. Annars behöver uttolkaren av informationen förstå skillnaden mellan leveransspecifikation och datamängd.



Figur 12 - Användandet av relationer för version och källa mellan olika dcat:Dataset i SmartFlow Server

2.2.5.4 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.



Figur 13 – dcat:Dataset och dcat:Distribution med ett urval av egenskaper

Ett exempel. En datamängd i form av en IFC 3D-modell levereras till SmartFlow Server för en redan registrerad leveransspecifikation. Data konverteras till RDF och lagras i en graf i GraphDB. 3D-modellen konverteras också till ett filformat för att få möjlighet att visualisera 3D-modellen i SmartFlow Server. 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 filen för 3D-visualisering
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

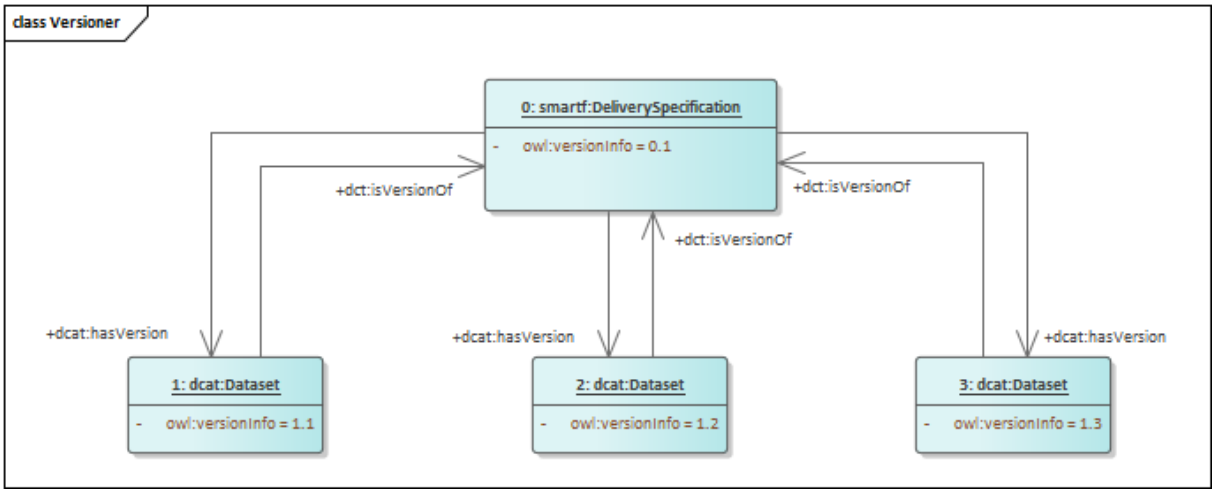
Precis som för leveransspecifikationer kan datamängder versionshanteras med hjälp av DCAT. Vi har sett en datamängd som en version av den leveransspecifikation den uppfyller vilket ger spårbarhet mellan datamängd och specifikation. Vi har valt en versionsnumrering med mönstret X.Y där X är versionen 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. Se Figur 14.

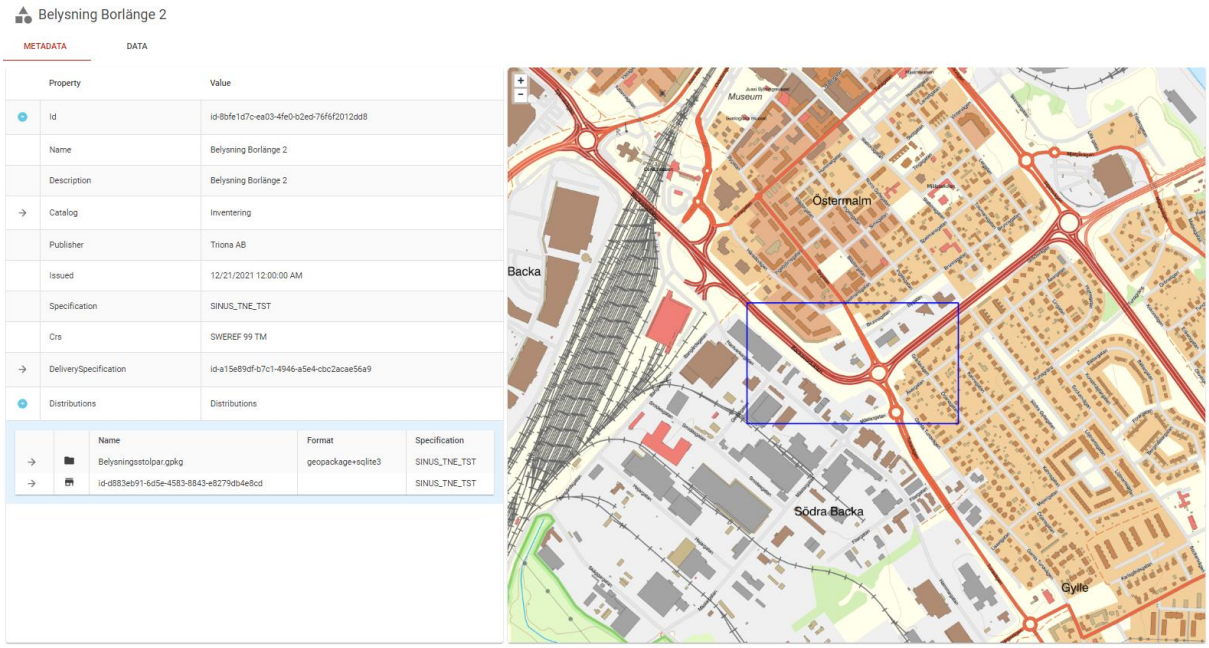
Metadata i aktuellt kapitel beskriver hela datamängder, inte enskilda objekt i en instansdatamängd. Det innebär att versionshanteringen ligger på hela datamängden, inte på enskilda objekt. En ny leverans av en konceptuell datamängd ger upphov till en ny version av den datamängden. Den nya versionen kan innehålla nya, borttagna och ändrade objekt beroende på vad man tillåter och definierar som en ny version. Versionering av objekt sker inom ramen för datamängdens specifikation innan data levereras till SmartFlow. Det vill säga, versionshantering av objekt ligger utanför SmartFlow Server.

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 12.

Se Figur 15 exempel på en beskrivning av en datamängd i SmartFlow Server.



Figur 14 - Olika versioner av samma konceptuella datamängd i SmartFlow Server



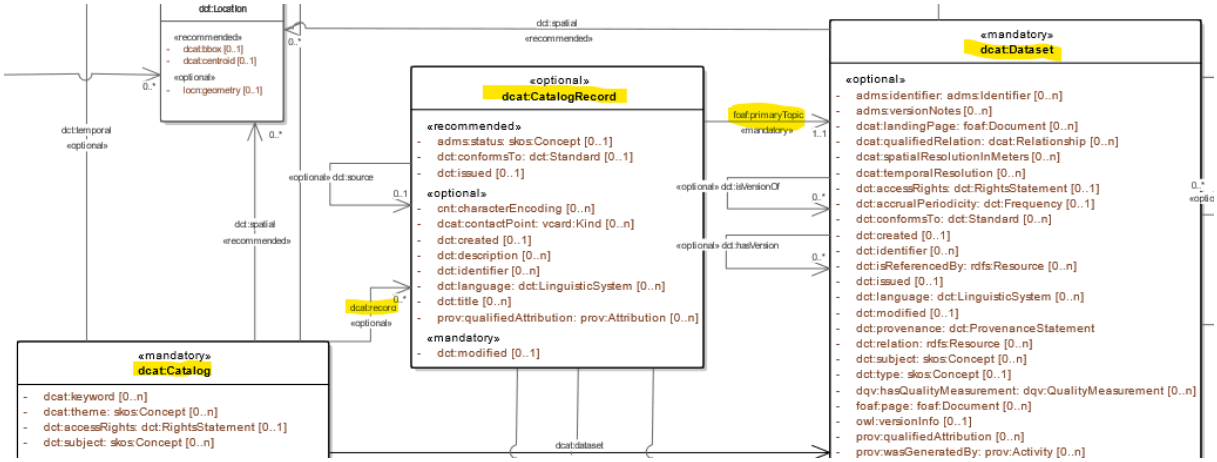
Figur 15 - Information om den levererade indatamängden Belysning Borlänge 2 (skärmbild från SmartFlow Server)

2.2.5.5 Händelser

Efter AP1 framgick det ett behov av att beskriva status eller händelser för de aktiviteter som ingår i flödet av data genom SmartFlow.

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

En händelse på en datamängd i en katalog representerades 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. En möjligen lite oortodox användning av dcat:CatalogRecord där man kanske kan förvänta sig att det borde vara dcat:CatalogRecord per dcat:Dataset; beroende på vad man anser att en dcat:CatalogRecord är.



Figur 16 - Ett utsnitt ur modellen för GeoDCAT som visar dcat:CatalogRecord och relationen till dcat:Catalog och dcat:Dataset

I AP3 gick vi inte vidare med händelser och användandet av dcat:CatalogRecord och implementationen av händelser i SmartFlow Server prioriterades ned.

Det finns fördelar med att representera en datamängd i en katalog med hjälp av dcat:CatalogRecord (dcat:Catalog har en relation till dcat:CatalogRecord via dcat:record och dcat:DatalogRecord har en reference till dcat:Dataset via foaf:primaryTopic) istället för en direkt relation mellan katalog och datamängd (via dcat:dataset), dvs att använda dcat:CatalogRecord som en slags proxy för ett dcat:Dataset:

1. dcat:CatalogRecord kan hålla titel och beskrivning på en datamängd även om man inte har tillgång till alla metadata om datamängden, vilket skulle kunna vara fallet i SmartFlow Server, se kapitel 2.2.6.
2. dcat:CatalogRecord har, via adms:status, möjlighet att beskriva ett aktuellt tillstånd för en datamängd

dcat:CatalogRecord finns inte beskriven i DCAT-AP-SE.

2.2.5.6 Egenskaper

Projektet har värderat och nyttjat ett fåtal av alla egenskaper i GeoDCAT-AP. Nedan följer några intressanta exempel.

Geografisk utbredning

Data, tänkt att levereras till och konsumeras från SmartFlow Server, är inom SmartFlow-projektet av geografisk natur (dock inte ett måste). Då är det också naturligt att beskriva dess geografiska egenskaper. En sådan egenskap är en datamängds geografiska utbredning. Genom att beskriva geografisk utbredning för datamängder är det möjligt att visualisera var datamängder finns geografiskt, söka efter datamängder geografisk och validera datamängder med avseende på geografisk utbredning.

DCAT ger möjlighet att beskriva geografisk utbredning både för kataloger och datamängder.

Vi har valt att beskriva geografisk utbredning i ett, per SmartFlow Server-instans, gemensamt spatialt referenssystem oavsett vilket referenssystem data har. Det förenklar visualisering, sökning och validering då ingen momentan spatial transformation behöver ske.

Geografisk utbredning för en datamängd räknas fram i SmartFlow Server baserat på den geografiska informationen i instansdatamängden. Det innebär att en spatial transformation av den geografiska utbredningen har implementerats i demo-versionen av SmartFlow Server då geografiska data i testfallen i flera fall har andra spatiala referenssystem än det gemensamma beskrivet ovan.

Den geografiska utbredningen representeras i DCAT av typen geosparql:wktLiteral, vilken förutom geometrin innehåller information om geometrins spatiala referenssystem.

Specifikation och format

Då de centrala uppgifterna för SmartFlow Server är att transformera data mellan olika specifikationer och konvertera data mellan olika format är det av yttersta vikt att beskriva specifikation för en datamängd och format på dess distributioner.

Det har vi huvudsakligen gjort via dct:conformsTo för dcat:Dataset samt dcat:mediaType för dcat:Distribution.

Spatialt referenssystem

Det spatiala referenssystem för den geografiska information hos en instansdatamängd representeras i metadata (inkl leveransspecifikationen), i RDF enligt GeoDCAT-AP, av en trippel av typen

> *dcat:Dataset dct:conformsTo dct:Standard*

där GeoDCAT-AP har följande kommentar till objektet:

> *Spatial reference systems SHOULD be specified by using the corresponding URIs from the "EPSG coordinate reference systems" register operated by OGC [OGC-EPSG].*

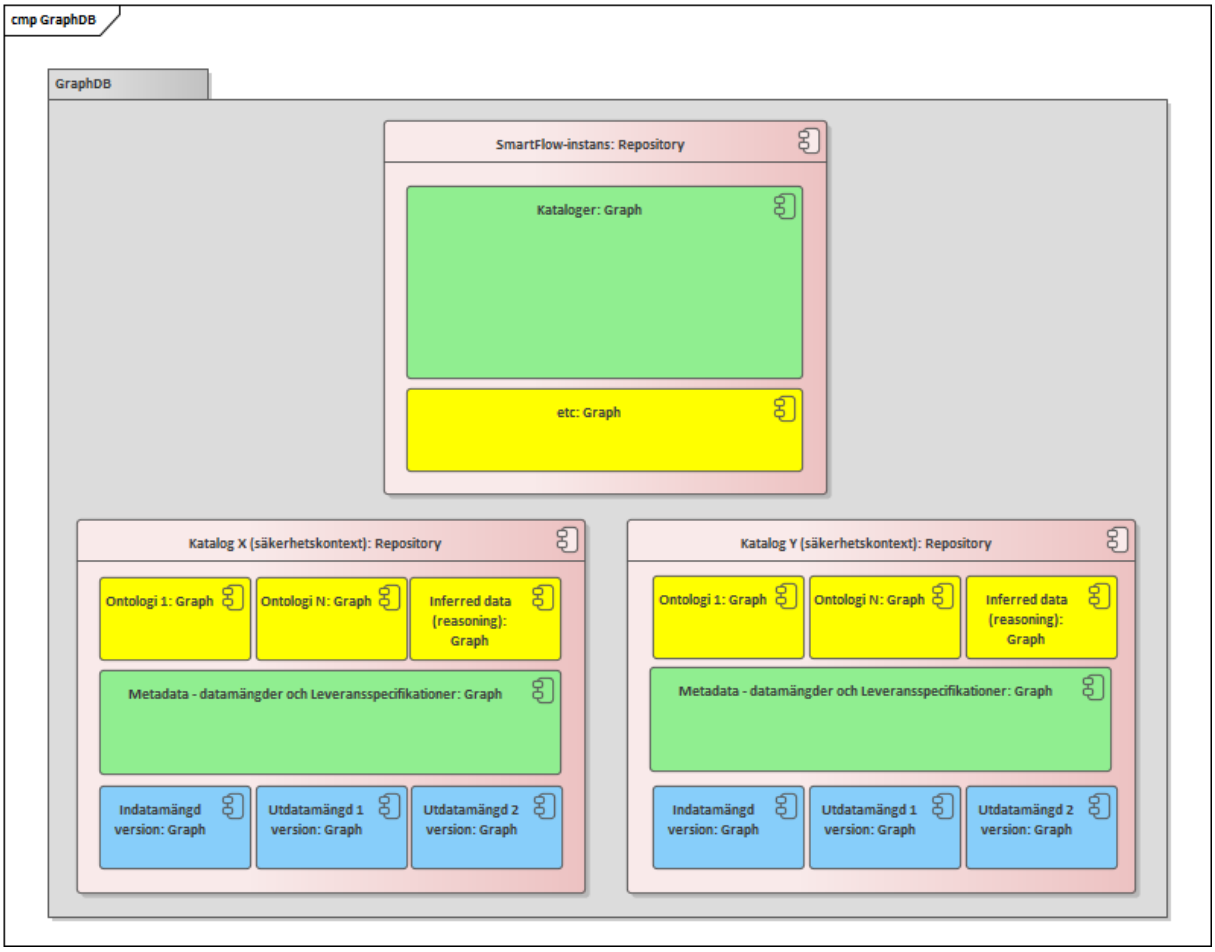
Olika spatiala referenssystem hos en leveransspecifikation för en indatamängd respektive en utdatamängd utgör grunden för/beskrivningen av en spatial transformation.

I demo-versionen av SmartFlow Server finns ingen spatial transformation av den geografiska informationen hos instansdatamängder implementerad men skulle i ett skarpt system mycket troligt utgöras av ett transformationssteg implementerat i programkod.

Det är ett troligt scenario då många indatamängder produceras i lokala eller regionala referenssystem men flera förvaltningssystem (konsumenter till SmartFlow Server) vill ha datamängder i nationella referenssystem.

2.2.6 Struktur i GraphDB

Vi har kommit fram till följande grundläggande struktur i GraphDB



Figur 17 - Struktur i GraphDB

Referenser till Figur 17 i texten inom parentes på formen graph@repository.

För varje instans av SmartFlow Server finns ett repository med en graf som innehåller kataloger och referenser till leveransspecifikationer och datamängder i dessa kataloger (*Kataloger@SmartFlow-instans*). Kodlistor och övrig systeminformation läggs i olika grafer i samma repository(*etc@SmartFlow-instans där etc representerar olika grafer*).

Metadatamängder och instansdatamängder läggs i separat repository enligt strukturen nedan.

- Ontologier för resonering läggs i en graf per ontologi (*Ontologi 1-N@Katalog X/Y*).
- De data som GraphDB själv kan härleda hamnar automatiskt i den graf som av GraphDB definierats för detta (*Inferred data@Katalog X/Y*).
- Metadata läggs i separat graf. Metadata gäller hela repositoryt (*Metadata@Katalog X/Y*).
- Varje importerad fil (instansdatamängd) läggs i separat graf. Varje graf motsvarar en version av filen (i metadata hålls flera versioner av samma datamängd samman) (*Indatamängd@Katalog X/Y*)

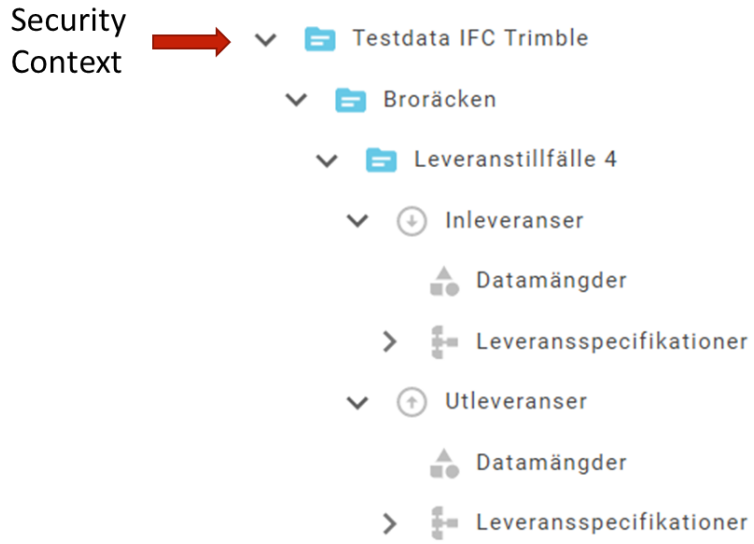
- De data som genereras av SmartFlow för konsumtion läggs i separata grafer som via metadata kopplas som separata distributioner av datasetet (*Utdatamängd 1/2@Katalog X/Y*).

Varje sådant repository motsvarar en katalog (dvs metadatamängder och instansdatamängder i den katalogen) och utgör då ett säkerhetskontext. Ett säkerhetskontext innebär ett sammanhang inom vilket datamängder värderas lika ur ett säkerhetsperspektiv, exempelvis utifrån sekretess eller tillgänglighet för olika användare/användargrupper.

En anledning till denna uppdelning är att GraphDB implementerar säkerhet på repository-nivå.

Beroende på säkerhetsvärderingen av data i SmartFlow Server kan en katalog (exempelvis ett projekt eller ett delprojekt) utgöra ett eget säkerhetskontext. Då kommer metadatamängder och instansdatamängder i den katalogen och dess underkataloger att sparas i ett eget repository.

För en datamängd gäller att den tillhör det säkerhetskontext som definierats för en katalog närmast uppåt i den kataloghirarki som datamängden är registrerad i.



Figur 18 - Exempel på kataloghierarki där katalogen Testdata IFC Trimble definierats som ett säkerhetskontext

Som minimum finns ett repository för metadatamängder och instansdatamängder per SmartFlow Server-instans. Maximalt finns ett repository per katalog som kan/ska innehålla datamängder.

2.2.7 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[1]-geometri, ofta IfcFacetedBrep där objektets avgränsning beskrivs av en mängd 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, eftersom RDF-strukturen innebär att data bryts ner i sina minsta beståndsdelar, kan resultera i 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 hanterat detta på flera sätt:

- För varje IFC-leverans så har Trimble levererat två filer:
    - En IFC-fil med all objektinformation men förenklad geometri (t ex som punkter och polylines)
    - En trb-fil (TrimBIM) för 3D-visning med Trimble Web3D. Denna fil innehåller identiteter (GUID) för objekten (samma GUID som i IFC-filen) men fullständig 3D-geometri för visualisering.
- Vi har med hjälp av SPARQL genererat GeoSPARQL-geometri och därigenom gjort förenkling inom ramen för GraphDB-miljön. Denna GeoSPARQL-geometri har sedan skickats vidare till en kartserver för effektiv kartvisning.

En annan aspekt på detta är att de förvaltningssystem som används för trafiknät ofta beskriver läget för objekt, händelser och liknande genom att man knyter läget till en position i ett nätverk. Nätverket, t ex väg- eller järnvägsnätet, beskrivs i detta fall som ett topologiskt nätverk, i grunden bestående av länkar som är förbundna med noder där det går att göra vägval. Länkarna beskrivs typiskt geometriskt med polylinjer och noderna med punkter. Lägen för objekt i förhållande till detta görs oftast genom att man beskriver längs vilken länk och var längs länken objektet finns. I många fall är man alltså endast intresserad av en punkt eller en sträcka längs en länk där objektet finns. Detta är också ett enkelt sätt att samköra läget för objekt som kommer från BIM och objekt som tillkommer på ett annat vis, t ex trafikdata, data om vägbeläggning eller olycksdata. Detta faktum innebär att komplicerad 3D-geometri för BIM-objekt behöver översättas till ett läge relativt nätet på det sätt som beskrivits ovan.

Vi tror att den bästa ansatsen är att i största möjliga mån utföra geometriförenkling innan data läses in i GraphDB. Om en fil med miljontals tripplar läses in är risken att prestanda påverkas negativt i hög grad utan någon egentlig nytta.

SmartFlow-projektet är inte det första som berörts av denna problematik. Exempel på arbeten och studier på området finns i nedanstående länkar.

- A method for converting IFC geometric data into GeoSPARQL
- Building Information Modeling

[1] BRep står för Boundary Representation och är en representation av 3D-objekt som definierar de geometriska avgränsningarna (boundaries) för objektet.

- An Improved Approach for Effective Describing Geometric Data in ifcOWL through WKT High Order Expressions
- Enhancing the ifcOWL ontology with an alternative representation for geometric data
- SimpleBIM: From full ifcOWL graphs to simplified building graphs

3 Lärdomar

3.1 Teknik

3.1.1 Länkade data

Länkade data och semantisk web är i dagsläget inte ”mainstream” inom det område som omfattas av SmartFlow-projektet, dvs information om den byggda miljön. Trots det borde det finnas stora möjligheter om dessa teknologier kan användas på rätt sätt, t ex som ett sätt att integrera data och dess beskrivningar från olika källor. Verksamheten är komplex med en mängd olika discipliner och standarder inblandade för att få en komplett beskrivning av t ex en infrastrukturanläggning. Arbete kring länkade data för den byggda miljön pågår eller har pågått, t ex internationellt i W3C, Linked Building Data Community Group, i Europa med Interlink och dess efterföljande CEDR call, i Sverige med RealEstateCore, i Norge med maskinlesbar V440 och på andra håll. En Google-sökning ger många svar, även om användningen ännu inte kan sägas ha nått ”industriell skala”.

Med SmartFlow har vi visat att länkade data och semantisk web kan användas för den användning som testats i projektet, dvs som en inleveransplattform för anläggningsdata av infrastrukturägare.

- Teknologin fungerar bra för automatiska översättningar mellan olika standarder, vokabulärer och strukturer/informationsmodeller, dvs för översättningar mellan olika informationsstrukturer, till exempel mellan IFC och den interna strukturen i ett förvaltningssystem.
- Teknologin medger att begreppssamband i hög grad kan beskrivas där domänkunskapen finns, dvs i verksamheten, i stället för att “gömmas” i olika informationssilo-system. Detta gör att dessa samband görs synliga för verksamheten vilket bör underlätta för denna att ta ägarskap över sina data
- För att fungera krävs ingående kunskaper kring de standarder som används för dataleveranser och de informationsstrukturer som används i en specifik verksamhet. Detta är dock ingenting nytt under solen. Skillnaden är sättet som dessa kunskaper görs tillgängliga och kan användas.
- Stora datamängder i kombination med omfattande och komplexa datastrukturer (scheman) innebär påfrestningar för prestanda i denna typ av lösning. IFC är ett exempel på en omfattande och komplex datastruktur. I detta fall kan man överväga att använda fokuserade delar av IFC-schemat (MVD:er) som grund istället för det kompletta schemat.
- Sökbara metadata utgör en central komponent för ett system som SmartFlow. Att beskriva förväntade och faktiska leveranser i form av krav och andra egenskaper utgör en grund för ett kvalitetssäkrat arbetssätt. I projektet har vi använt ontologin GeoDCAT-AP för detta men det finns även andra ontologier som kan vara relevanta.
    - Som metadata ingår även de specifikationer (maskinläsbara) som ligger till grund för maskinell verifiering av levererade data. Exempel på maskinläsbara specifikationer kan vara EXPRESS-scheman (som för IFC), MVD:er, XML-scheman eller OWL-ontologier tillsammans med SHACL-regler.

3.1.2 CoClass

CoClass beskrivs som ”samhällsbyggarnas gemensamma klassifikationssystem för framtidssäkrad dokumentation”. Man kan säga att CoClass definierar ett gemensamt språk för den byggda miljön. Ett gemensamt språk är naturligtvis en förutsättning för en lyckad och meningsfull kommunikation. I SmartFlow-projektet har vi använt CoClass som ett gemensamt språk för att beskriva data, oavsett

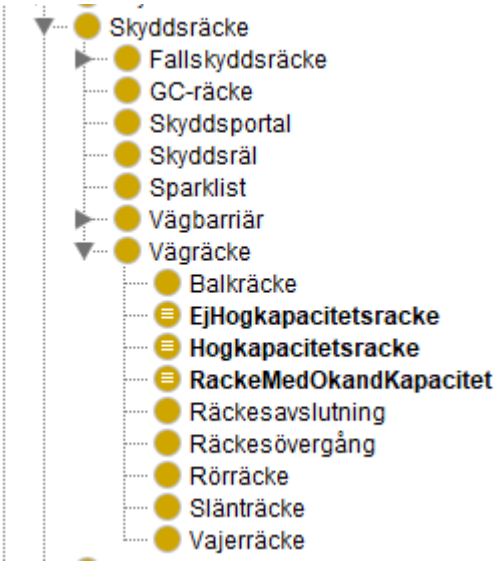
datas ursprung eller destination. På detta sätt förenklas de översättningar som behöver göras i och med att antalet översättningar minimeras.

Tyvärr finns inte CoClass tillgängligt som länkade data i dagsläget. Dock visade det sig enkelt att, för våra behov i SmartFlow, göra en maskinell översättning från CoClass egna struktur. Hur detta har gjorts beskrivs i kapitel 2.2.1.

En stor fördel med CoClass är, som tidigare nämnts, att grundstrukturen i CoClass definierar en taxonomi av begrepp utan att blanda in applikationsspecifika egenskapskrav i definitionerna. Detta medför att möjligheten till en bred återanvändning ökar markant eftersom egenskapskraven överlåts till kravställningen på leveranserna och kravställningen från olika tillämpningar. På det sättet så har vi bedömt att CoClass är mycket lämpligt för användning som kanonisk modell för SmartFlow.

För att få till en användning som fungerar på detaljnivå så behöver i många fall en specialisering av begreppen i CoClass göras. Detta har vi gjort genom att ta fram en länkad ontologi specifikt för de testfall som genomförts som tillför de specialiserade klass- och egenskapsbegrepp som krävts utan att i sig påverka CoClass.

Ett exempel på klassnivå rör klassen Vägräcke i CoClass. Datakatalogen i STVDB har en egenskap för vägräcke som rör huruvida räcket är ett högkapacitetsräcke eller ej. För att lösa det i OWL så har subklasser till CoClass Vägräcke lagts till i Trafikverkets ontologi enligt nedanstående bild.



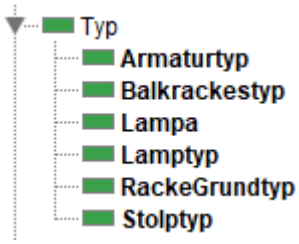
För att göra definitionerna explicita så används egenskapen kapacitetsklass enligt nedanstående exempel för Högkapacitetsräcke.

Informationen har hämtats från https://www.nordicroadsafety.com/regelverk/vag-och-broracke/en-1317-2/, nedanstående tabell.

Kapacitetsklass	
Lågkapcitet	T1
	T2
	T3
Normal kapacitet	N1
	N2
Hög kapacitet	H1
	L1
	H2
	L2
	H3
	L3
Mycket hög kapacitet	H4a
	H4b
	L4a
	L4b

Definitionen innebär att alla objekt som har en kapacitetsklass med värdet H1, H2, H3, H4a och H4b automatiskt klassas som Högkapacitetsräcke.

När det gäller egenskaper så behöver ibland mera detaljerade egenskaper än de som hittas I CoClass läggas till för att åstadkomma en entydig begreppsmappning. Ett exempel är CoClass-egenskapen Typ där man för att exempelvis kunna knyta fördefinierade kodlistor för armaturer, lampor, stolpar etc behöver specifika egenskapstyper för dessa. Genom att definiera dessa sub-egenskaper kan man överföra specifika egenskaper för objekt men fortfarande söka efter data med hjälp av egenskapstypen Typ för t ex en Armatur.



3.1.3 IFC-standarden

Det ter sig ganska tydligt att, när det gäller utbyte av information kring den fysiska anläggningen speciellt vid nyinvestering, så kommer IFC i och med version 4.3 och kommande versioner att spela en tung roll. Dock gäller det att komma ihåg att det finns ett överhängande behov att samla in information om den anläggning som redan finns i drift men där data saknas helt eller delvis. När det gäller annan typ av anläggningsinformation, t ex signal för järnväg, trafikdata etc så kommer andra standarder troligen att användas.

- IFC-standarden finns utgiven som IfcOWL vilket ger goda förutsättningar för användning i SmartFlow. Dessutom finns färdiga komponenter tillgängliga under Open Source-licens för konvertering av IFC-data på STEP-format till RDF-format enligt IfcOWL. 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.

- IFC är ett exempel på en omfattande och komplex datastruktur. I detta fall kan man överväga att använda fokuserade delar av IFC-schemat (MVD:er) som grund istället för det kompletta schemat om eller när prestanda är avgörande.
- Det är i skrivande stund oklart vad som utgör den bästa ansatsen för verifiering av IFC- och andra typer leveranser.
    - MVDXml finns att tillgå, men hur kan dessa tas fram på bästa sätt? Hur är tillgången på programvara som kan utföra maskinell verifiering baserat på MVDXml? MVDXml är också anpassat specifikt för IFC. För data som inte ryms i IFC-världen gäller det att hitta andra sätt för maskinell verifiering.
    - Det pågår utveckling inom BuildingSMART med IDS (Information Delivery Specification). Specifikationen beskrivs på följande sätt:
        - a computer interpretable document that defines the Exchange Requirements of model based exchange. It defines how objects, classifications, properties, and even values and units need to be delivered and exchanged. This can be a combination of Industry Foundation Classes (IFC), Domain Extensions, and additional classifications and properties (national agreements or company specific ones; either stored in bSDD or somewhere else).
        - I skrivande stund är det oklart vad det är för status på specifikationen och huruvida det finns mjukvara som kan tolka en specifikation för att maskinellt verifiera leveranser. Om specifikationen blir framgångsrik och funktionell kan detta självklart vara ett alternativ, speciellt om specifikationen tillåter verifiering av data även utanför IFC-domänen.
    - En möjlig ansats kan vara att bygga verifiering baserat på SHACL. Det kan även vara en möjlighet att generera SHACL utifrån MVDXml. SHACL har fördelen att man då utför kontroller mot RDF-data vilket möjliggör verifiering av data som har sitt ursprung utanför IFC. Detta behöver dock testas mera
- En annan påfrestning för denna typ av lösning är den många gånger komplexa och omfattande geometribeskrivning som förekommer när det rör sig om 3D-representation av komplexa objekt. Vi ser det inte som meningsfullt i dagens läge att använda länkade data och semantisk web för att bearbeta denna typ av geometri. Detta gör att följande slutsatser kan dras:
    - Fullständig 3D-geometri bör tas omhand separat i miljöer som klarar att lagra, visualisera och på andra sätt bearbeta och analysera denna typ av geometri.
    - Fullständig geometri bör lagras på sådant sätt att den enkelt kan länkas till de objekt som bär geometrin. OGC GeoSPARQL är en standard på GIS-området där man tagit fram lösningar för hur man kan länka samman geometriska beskrivningar med de objekt som använder dessa beskrivningar
- För hantering av de flöden som normalt förekommer mellan nyinvesteringsprojekt och förvaltning finns det normalt ett behov att förenkla geometrin så att avancerad 3D-geometri

generaliseras som punkter, linjer och ytor för att ge information om det geografiska läget för en tillgång. Denna typ av enkel geometrisk representation är den som normalt används vid inventering, t ex via fotografering eller laserscanning.

- Inom långsträckt infrastruktur finns oftast ett behov att beskriva tillgångarnas läge relativt ett nätverk för att enkelt kunna samköras med annan information, t ex trafikmängder, trafikrestriktioner och –regler, trafikolyckor etc. Därför behöver den generaliserade geometrin ofta även översättas till ett läge relativt nätverket (väg- eller järnvägsnätet). Hur dessa lägen kan representeras beskrivs bland annat i SS637004 eller Nasjonal Vegdatabank.

3.1.4 Filformat för inventering

Vi känner inte till att det finns färdiga standarder att använda för dessa typer av data. De manuella verktyg som används av Trafikverket levererar någon typ av tabellformat med attribut plus geometrier, t ex shape-format, OGC GeoPackage eller ESRI geodatabas. Detsamma verkar gälla om data kommer från icke-manuella källor såsom lasermoln eller 360-bilder.

Gemensamt för dessa format är att man i grunden kan betrakta dom som bestående av tabelldata där en tabell beskriver förekomster (instanser) för en given objekttyp och där varje kolumn beskriver en given egenskap. Denna typ av data låter sig enkelt konverteras till RDF på följande sätt:

- Varje tabell motsvarar en owl:Class
- Varje kolumn motsvarar ett owl:DatatypeProperty, dvs predikat i en trippel.
- Varje rad motsvarar ett subjekt. Någon princip behöver användas för att generera en URI för subjektet.
- Varje cell motsvarar ett objekt i en trippel

3.1.5 Publicerade ontologier

Det är naturligtvis en fördel att återanvända befintliga, redan publicerade, ontologier där så är möjligt. Ju större spridning ”ett språk” har desto större möjlighet finns att det finns verktygsstöd, kunnande etc att få tag på. Inom SmartFlow har vi kunnat använda:

- nvdb-owl som beskriver innehållet i norska NVDB
- IfcOWL som beskriver IFC
- DCAT som beskriver metadata (enligt profilen GeoDCAT-AP)

Flera relevanta ontologier finns naturligtvis att tillgå såsom SSN/SOSA för sensordata, PROV-O för proviniens, QUDT för kvantiteter och enheter etc.

CoClass, som beskrivits ovan, finns i dagsläget inte publicerad officiellt som OWL-ontologi. Dock finns alla förutsättningar för att göra det, vilket SmartFlow-projektet har visat genom den i projektet genererade ontologin.

När det gäller svenska NVDB och Trionas TNE-produkt så finns datakataloger och instansdata definierade på ett format som följer Svensk Standard (SS 637006). För dessa datakataloger och instansdata så har vi inom ramen för SmartFlow tagit fram översättningsfunktionalitet till RDF och OWL. Här skulle man kunna önska att denna representation fanns tillgänglig från Trafikverket på samma sätt som nvdb-owl i Norge.

3.1.6 Demo-applikation för SmartFlow

Demo-applikationen har i första hand tagits fram för att på ett begripligt sätt kunna demonstrera SmartFlow samt testa de tekniker och principer som presenterats i detta dokument. Vid en eventuell produktifiering behöver åtminstone följande punkter beaktas:

- Om en produkt ska utvecklas för kommersiellt syfte och användning hos flera så behöver applikationen i största möjliga mån göras oberoende av underliggande kart- och 3D-visning samt lagring triple store.
    - I fallet med triple store bör det vara relativt enkelt så länge produkten uppfyller specifikationerna SPARQL-protokollet. Dock kan olika produkter ha kommit olika långt när det gäller exempelvis stöd för reasoning (åtminstone enligt profilen OWL-RL), GeoSPARQL, SHACL och liknande. En produkt bör ha stöd för alla dessa för att vara aktuell för SmartFlow.
    - Karttjänster är idag standardiserade, så även här bör det vara relativt enkelt att göra en applikation oberoende av bakomliggande implementationer
    - När det gäller 3D-visning av modeller över web så har vi inte hittat några standardiserade protokoll för detta som även inkluderar någonting som motsvarar Web Feature Service (WFS) som finns på kartsidan. För en effektiv web-rendering av 3D-modeller som också ger möjlighet att länka 3D-objekt med data i triple store så har vi använt Trimble:s TrimBim-format och Trimbles WorldViewer.
- Om en produkt utvecklas för en specifik organisation så har organisationen ofta redan gjort val när det gäller lagring och presentation/visualisering, vilket gör situationen en aning mindre komplex.
- Demo-applikationen har en arkitektur som bygger på hög grad av modularisering och, via en meddelande-baserad arkitektur, låg grad av koppling mellan moduler. Detta tror vi är nödvändigt för en SmartFlow-lösning för att kunna erbjuda nödvändig flexibilitet och skalbarhet.
- Demo-applikationen har utvecklats av personer som inte har web-utveckling som specialitet. Därför behöver web-specifika delar troligen arbetas om i någon grad av web-utvecklare.

3.2 Människor och organisation

Teknologin är som tidigare nämnts i nuläget inte ”mainstream” inom området tillämpningar för bygg och anläggning. Teknologin innebär också att ansvaret för stringenta definitioner av begrepp och begreppssamband, som kanske tidigare funnits hos enskilda utvecklare av tillämpningar, nu flyttas närmare verksamheten. Detta kommer att innebära att människor som ska jobba med dessa definitioner kommer att behöva utbildning och bra verktyg att arbeta med.

En övergång från traditionella leveranser i form av ritningar och dokument till modellbaserade leveranser enligt nya standarder kommer att innebära att kravställningar kommer att förändras och göras på ett nytt sätt. Till exempel kommer antagligen kravställningen att behöva finnas på maskinläsbar form för att åstadkomma en möjlighet till maskinell dataverifiering för att minimera risker för fel pga ”den mänskliga faktorn”. Detta är ett arbete som initialt kommer att kosta pengar och energi och risken finns att det uppstår motstånd och diskussioner om vem som ska ta kostnaderna för detta.

Men det är sällan som det är teknologier i första hand som sätter käppar i hjulet. Det finns alltid ett inbyggt motstånd mot förändring av invanda arbetssätt, speciellt när det gäller arbetssätt som rör flera områden inom en organisation. Vår erfarenhet är att större förändringar behöver införas i små steg som en evolution i stället för revolution. En rekommendation för en eventuell fortsättning blir därför att börja smått inom ett begränsat område och att därifrån växa gradvis och strukturerat. Det skulle kunna innebära att man startar med en eller två verksamheter som får arbeta tillsammans för att utveckla konceptet vidare. Det brukar vara så att lyckade initiativ drar till sig uppmärksamhet och personer på ett sätt som inte skiljer sig avsevärt från gravitation. Ju större massa, desto större är gravitationen.

4 Vad händer efter SmartFlow projektets avslut

4.1.1 Informationsspridning

Projektets parter kommer fortsätta att sprida information om projektet och dess resultat. Det kan komma att ske via riktade föredrag samt vid konferenser. Det kan finnas en möjlighet att hålla föredrag vid kommande BuildingSMART-summit samt kommande LDAC-konferens (Linked Data in Architecture and Construction, senaste konferensen hölls i Oktober 2021)

4.1.2 Fördjupade tester

Målet är att under 2022 fortsätta med fördjupade tester av SmartFlow Server konceptet, primärt hos projektets behovsägare, för att få en bättre bild av hur projektets resultat bör utvecklas vidare.

4.1.3 Produktifiering av konceptet

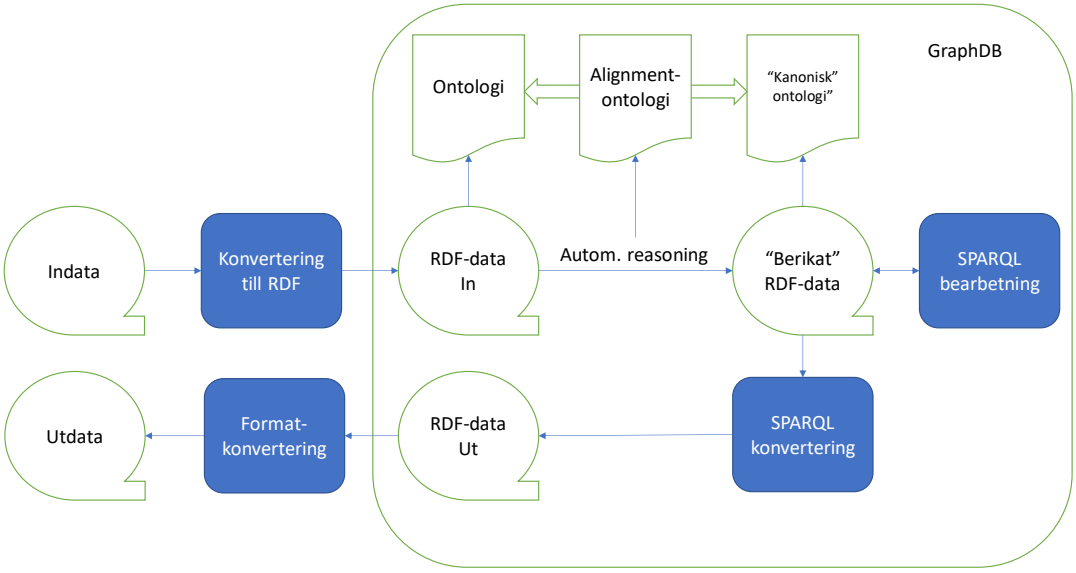
Det är ett mål att den konceptlösning som tagits fram inom projektet ska produktifieras och därefter erbjudas fler parter på marknaden. När detta kommer att ske får visa sig efter de fördjupade testerna och en marknadsanalys.

5 Appendix 1 – Beskrivning av typer av testfall dataflöden

5.1 Inledning

5.1.1 Bearbetning i GraphDB

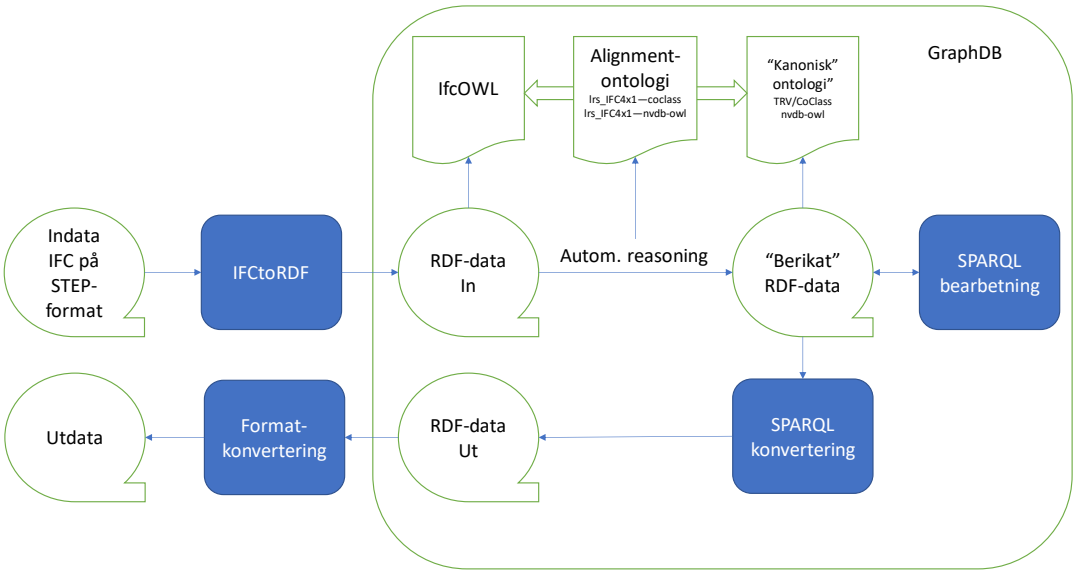
För att få kontroll på exakt vilken datamängd som utgör indata samt vilken datamängd som utgör utdata så har vi hanterat dessa mängder på följande sätt där varje datamängd utgör en separat graf i ett databas-repository:



Figur 19 - Generellt flöde för testfall

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). I stället kan vokabulären från den kanoniska ontologin alltid användas. Den kanoniska ontologin behöver därför vara väl känd hos de som använder miljön.
    b. Mappningen sker med alignment-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

5.2 Indata: IFC



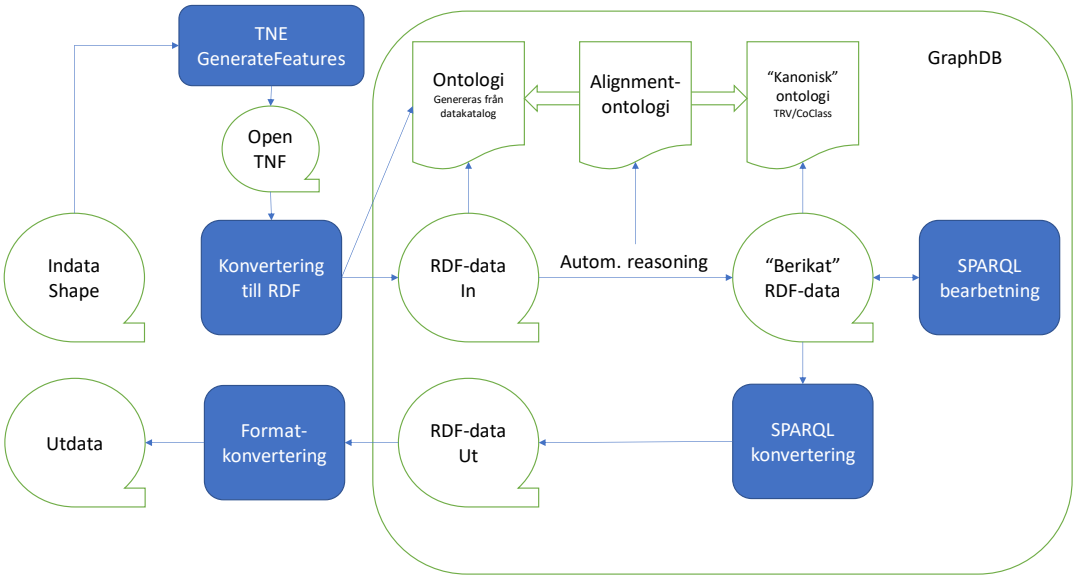
Figur 20 - Flöde för IFC-data

1. IFC-filer på STEP-format (ISO 10303-21) konverteras med hjälp av IFCtoRDF till RDF-format enligt ifcOWL.
2. RDF-data importeras till GraphDB
3. 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). I stället kan vokabulären från den kanoniska ontologin alltid användas. Den kanoniska ontologin behöver därför vara väl känd hos de som använder miljön.
    b. Mappningen sker med alignment-ontologier som länkar indataontologin mot den kanoniska ontologin.
    c. Egenskaper i property sets enligt IFC representeras endast som listor med kod/värde-par där kod och värde utgörs av text-strängar. Dessa properties översätts med hjälp av SPARQL till äkta RDF-properties (URI) för att kunna länkas via alignment-ontologier och därmed automatiskt översättas. För detta syfte har vi utvecklat en separat ontologi som med OWL beskriver dessa property sets. När processen är klar har alla properties från property sets översatts till RDF-properties och dessutom via reasoning översatts i termer av properties i den kanoniska ontologin.
4. Viss berikning av data kan ske ytterligare via specifika SPARQL-scripts.
    a. GeoSPARQL-geometrier (punkt, linje, yta) genereras med hjälp av SPARQL för objekten så att dessa kan visas på karta, överföras till kartserver, sökas ut spatialt samt matchas mot nät.
5. Specifika utdata görs via SPARQL-scripts där man har alla möjligheter att göra beräkningar, aggregeringar osv för att anpassa utdata mot specifika krav
    a. För attributvärden som kommer från IFC och är text-strängar men där mottagande system har uppräkningslistor, vilket ofta är fallet i NVDB hos Statens Vegvesen så behöver text-strängarna översättas till de URI:er som gäller i dessa uppräkningslistor. Detta görs med hjälp av SPARQL genom att matcha text-strängarna från property

sets mot motsvarande rdfs:label från mottagande systems ontologi. För närvarande, i våra testfall, behövs exakt matchning för att detta ska fungera.

6. Till sist kan utdata behöva konverteras till nytt format för leverans
    a. I fallet med NVDB hos Statens Vegvesen konverteras data till GML enligt fördefinierade XML-scheman.
    b. Om data ska levereras till NVDB eller motsvarande i Sverige så konverteras data till OpenTNF
    c. För GUS levereras för närvarande endast RDF. Dessa data skulle, vid behov, kunna konverteras till exempelvis Excel-format (eller csv).

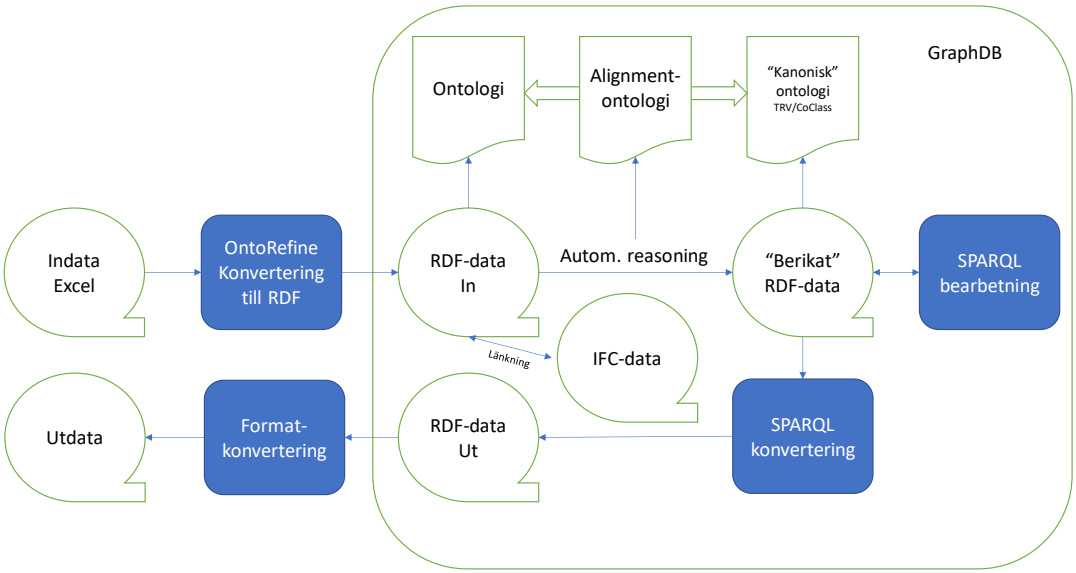
5.3 Indata: Shape



Figur 21 - Flöde shape-data

1. Shape-filer konverteras med hjälp av TNE GenerateFeature (en funktion i TNE-produkten för att skapa företeelsedata från t ex shape-filer) till OpenTNF.
2. Från OpenTNF-data genereras både ontologi (från datakatalogen som definierats manuellt via TNE Catalogue Editor) och instansdata på RDF-format.
3. RDF-data (både ontologi och instansdata) importeras till GraphDB
4. 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
5. Viss berikning av data kan ske ytterligare via specifika SPARQL-scripts.
    a. GeoSPARQL-geometrier (punkt, linje, yta) genereras med hjälp av SPARQL för objekten så att dessa kan visas på karta, överföras till kartserver, sökas ut spatialt samt matchas mot nät.
6. Specifika utdata görs via SPARQL-scripts där man har alla möjligheter att göra beräkningar, aggregeringar osv för att anpassa utdata mot specifika krav
7. Till sist kan utdata behöva konverteras till nytt format för leverans
    a. Om data ska levereras till NVDB eller motsvarande i Sverige så konverteras data till OpenTNF
    b. För GUS levereras för närvarande endast RDF. Dessa data skulle, vid behov, kunna konverteras till exempelvis Excel-format (eller csv).

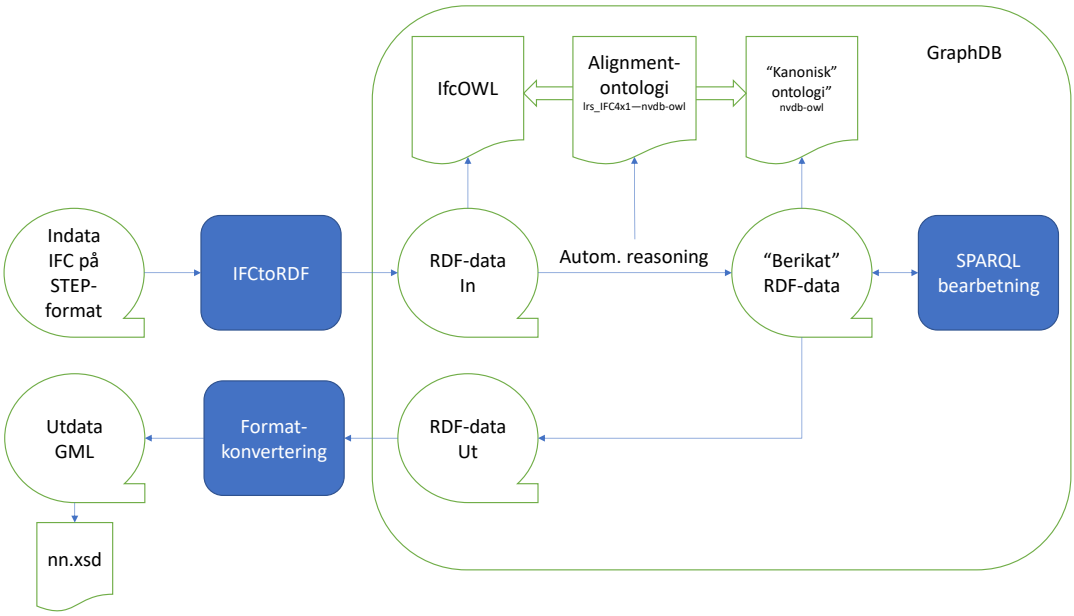
5.4 Indata: Excel



Figur 22 - Flöde excel-data

1. Excel-filer konverteras med hjälp av OntoRefine så att RDF genereras mot en manuellt definierad ontologi.
    a. Ontologin definieras så att ett excel-blad motsvarar en klass och varje kolumn ett property. Varje rad blir ett subjekt.
2. RDF-data (både ontologi och instansdata) importeras till GraphDB
3. 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
4. I excel-fallet är data kopplade mot IFC-objekt via ett komponent-id. Därför körs en process via SPARQL där excel-data och IFC-data i ett separat steg länkas samman
5. Efter excel-importen, i och med att data kopplats samman med IFC-data, följer processen samma steg som för IFC eftersom vi nu har kompletta IFC-objekt som även inkluderar egenskaper från excel och som hanteras som vilka andra egenskaper som helst.

5.5 Utdata: GML (NVDB-datakatalog)



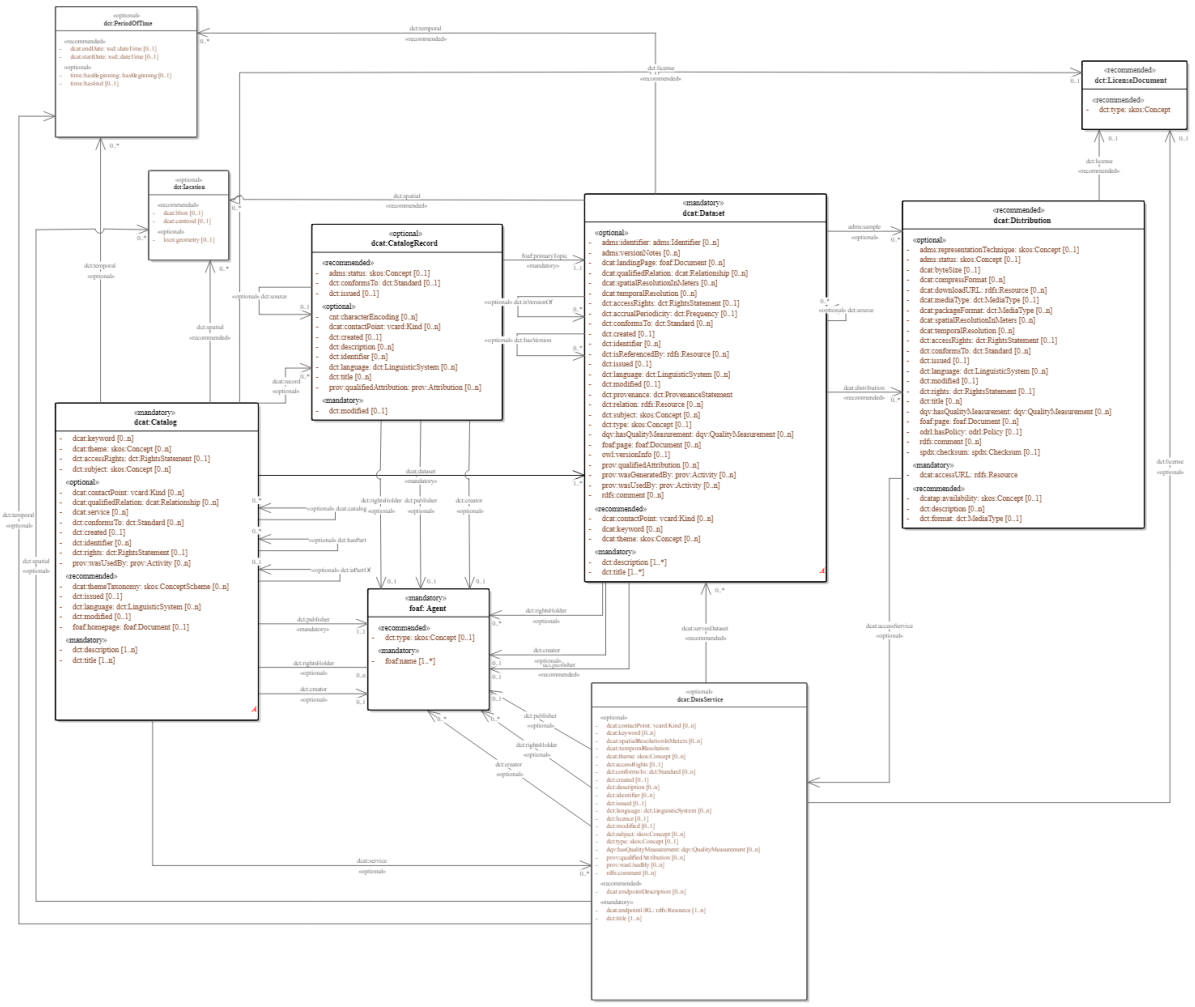
Figur 23 - Flöde utdata i GML

1. IFC-filer på STEP-format (ISO 10303-21) konverteras och importeras på samma sätt som kapitel 5.2
2. Utdata konverteras till XML-format enligt scheman som finns tillgängliga här: https://github.com/vegvesen/NVDB-Datakatalogen/tree/master/GML
    a. Idealt sett skulle denna formatkonvertering göras via generella komponenter som utför godtyckliga konverteringar RDF=>XML. Det finns sådana, t ex inom ramen för TopBraid Composer. Tyvärr går denna inte att anropa från egen kod. För att på snabbast möjliga sätt åstadkomma önskat resultat för demonstration så utvecklades en egen, icke generell, komponent med C# som utför konverteringar för utpekade företeelsetyper.

6 Appendix 2 - Metadata

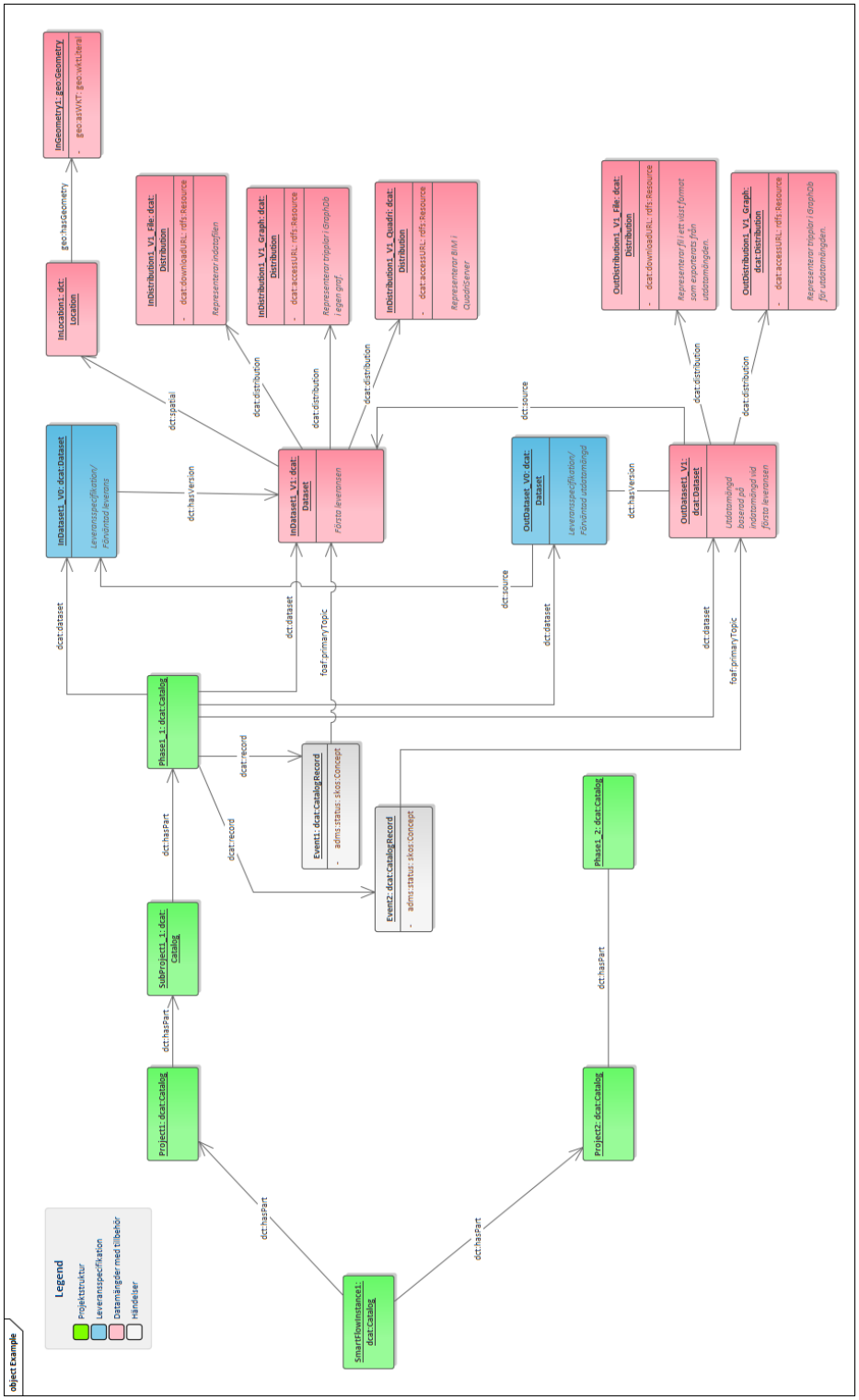
6.1 GeoDCAT-AP - Version 2.0.0

Nedan visas modellen för GeoDCAT version 2.0.0 så som den är publicerad i [3].



6.2 Metadatastruktur, exempel

Modellen nedan visar ett generellt exempel på hur metadata har strukturerats med hjälp av DCAT i SmartFlow.



7 Appendix 3 – Använda ontologier

7.1 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”

7.2 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.

7.3 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.