2015

Onderzoeksrapport

DATA INTEGRATION FRAMEWORK

DOMINIK KEPINSKI

HZ UNIVERSITY OF APPLIED SCIENCES

Afstudeerscriptie van Dominik Kepinski

Studentnummer 61535

In het kader van de opleiding HBO-ICT aan de

HZ University of Applied Sciences, Vlissingen

Versiebeheer

Versie	Auteur	Omschrijving
0.1	D.K.	Onderzoeksrapport, informele concept
0.2	D.K.	Onderzoeksrapport, eerste concept versie
0.3	D.K.	Onderzoeksrapport, tweede concept versie
0.4	D.K.	Groenlicht versie
0.5	D.K.	Verwerken feedback groenlicht gesprek
1.0	D.K.	Final

Samenvatting

Het lectoraat DIO/EMM doet een onderzoek naar de mogelijkheden om een semantisch web toe te passen als hulpmiddel bij kennisuitwisseling. Voor dit doel zijn er diverse databronnen beschikbaar gemaakt, zoals sensoren, databases en web-services. Het probleem ligt in de diverse natuur van deze databronnen en in het continu genereren van nieuwe data(soorten) door de onderliggende systemen. Daarom is er een behoefte ontstaan aan een data integration framework, waarmee niet-semantische data zonder vaste structuur zodanig getransformeerd worden, dat deze in een semantische database (triplestore) opgenomen en geanalyseerd kunnen worden.

Om dit doel te realiseren is er een onderzoek gedaan naar de achterliggende theorie. Daarin is onderzocht hoe het semantische web werkt, en wat er al bekend is op het gebied van data integration binnen het semantische web. Vervolgens is er met de stakeholders besproken wat er van een data integration framework verwacht wordt. Op basis van deze gesprekken zijn er een aantal potentiële problemen in kaart gebracht, en er is een lijst van requirements opgesteld. Op basis van de theorie en de eerder opgestelde lijst van requirements is er een ontwerp opgesteld. Dit ontwerp is vervolgens omgezet naar een prototype. Dit prototype was opgebouwd op basis van al bestaande bedrijfsprocessen. Op basis hiervan zijn er een aantal tests uitgevoerd, waaronder performance- en accuracy-test.

Op basis van het theoretisch onderzoek bleek dat er al groepen onderzoekers bezig waren met deze kwesties. Zo was er ook een aantal tools - dan wel in een vroeg stadium van development – dat hulp kan bieden aan de - binnen het lectoraat DIO aanwezige - problemen. Uit een lijst van potentiële oplossingen, is er gekozen voor OpenRefine en RML-Processor. Deze keuze is gebaseerd op de opgestelde lijst van requirements, waaronder het systematisch en deels automatisch opschonen van grote datasets, en vervolgens het vertalen daarvan naar semantische data, om op ten slotte in een triple store ingeladen te worden. Uit de tests bleek, dat combinatie van OpenRefine en RML-Processor een positieve bijdrage kan leveren aan de bedrijfsprocessen binnen het lectoraat DIO. Deze toepassing zorgt ervoor dat de hoeveelheid werk die handmatig uitgevoerd moet worden, beperkt wordt tot het eenmalig opstellen van een ontwerp/mapping, en het uitvoeren hiervan. Dat is veel efficiënter vergeleken met de huidige aanpak, waarbij deze bewerkingen handmatig uitgevoerd worden.

Summary

The lectureship DIO/EMM is researching the possible application of semantic web as aid in the process of knowledge exchange. For this purpose, multiple data sources have been made available, for example sensors, databases and web-services. The problem stems from the diverse nature of said sources and the continuous generation of new data (types) produced by the sub-systems. This has caused a need for a data integration framework, which would transform the non-semantic data so that it can be integrated into a semantic database (triplestore) and analyzed.

To that end a research has been carried out on the available literary theory. There, the workings of the semantic web have been analyzed, and the currently available knowledge of semantic data integration has been collected. Afterwards, the relevant stakeholders have been interviewed, to find out their expectations of a data integration framework. Based on those interviews the potential problems have been mapped and a requirements list has been formulated. Based on the gathered knowledge and the aforementioned requirements list, a design has been created. Said design formed a base for the development of a prototype. This prototype was used in a multitude of tests and experiments, to assess performance and accuracy.

Based on the findings of the theoretical research, it became apparent that a multitude of research groups have performed similar inquiries. Also, a number of tools – despite being in early development – present a solution to the problems found in the lectureship DIO. From the list of potential solution, OpenRefine and RML Processor have been chosen. This choice is based on the requirements, which included the systematic and partially automatic cleaning of large data sources, and the transformation into semantic data, so that it may be loaded in a triplestore. From the experiments perfumed on the prototype, it became apparent that OpenRefine and RML Processor could have a positive impact on the business processes of the lectureship DIO. This application reduces the amount of work that has to be performed manually, down to the designing a mapping, then running it. This is streamlined compared to the previous approach, where said actions had to be performed manually each time.

Voorwoord

Tijdens mijn afstudeerperiode heb ik ontzettend veel kunnen leren over technologieën die voor mij nog onbekend waren. De opdracht was af en toe een beetje overweldigend, maar gelukkig heb ik daarbij hulp gehad. Daarom wil ik graag mijn bedrijfsbegeleider Hans de Bruin en mijn afstudeerdocent Daan de Waard bedanken voor de begeleiding gedurende dit project. De feedback, kritische vragen en adviezen hebben ertoe geleid dat de kwaliteit van het onderzoek en het rapport groter werd.

Ook wil ik graag mevrouw Petra Romijn-Verschoof bedanken voor de taalkundige hulp bij het opstellen van dit rapport. Er is ontzettend veel tijd en moeite ingestopt, om het grammaticaal gedeelte van het rapport te verbeteren.

Daarnaast wil ik de medewerkers van het lectoraat DIO bedanken voor de leuke discussies die plaatsvonden, waardoor er bij mij meer inzicht is ontstaan over de kwesties van ICT en het semantische web.

Ik wens u veel leesplezier!

Dominik Kepinski

Vlissingen, 02-07-2015

Inhoud

1 Inleiding

In het kader van het afstuderen van de opleiding ICT is een onderzoeksvoorstel opgesteld door Dominik Kepinski, student ICT aan de HZ University of Applied Sciences.

In dit document wordt beschreven met welke methoden en activiteiten er een antwoord gegeven zal worden op de gestelde hoofdvraag.

In dit hoofdstuk wordt het onderzoek in hoofdlijnen beschreven. Zo zal de organisatie en de aanleiding tot onderzoek kort beschreven worden. Vervolgens wordt de probleemstelling gegeven en hieruit volgt de hoofdvraag. Deze hoofdvraag wordt vervolgens opgedeeld in kleinere deelvragen, die gezamenlijk een antwoord zullen geven op de hoofdvraag.

Het onderzoek wordt uitgevoerd in opdracht van Hans de Bruin, lector DIO (Duurzaam Innoveren en Ondernemen) aan de HZ University of Applied Sciences. Het lectoraat DIO houdt zich op dit moment bezig met kennismanagement volgens Expertise Management Method (EMM). Deze methode is door Hans de Bruin ontwikkeld en wordt o.a. toegepast door Zeeweringen en DeltaExpertise. Deze methode zorgt ervoor dat kennis en “knowhow” vastgelegd worden, zodat deze uitgewisseld kunnen worden in de beroepspraktijk maar ook in het onderwijs. Het uiteindelijke doel van deze methode is valorisatie oftewel, het verrijken van kennis en data zodat deze meerwaarde heeft.

1.1 Aanleiding

Op het gebied van EMM zijn er door de studenten en de medewerkers van het lectoraat onderzoeken uitgevoerd. Tijdens deze onderzoeken is er naar voren gekomen dat de opslag van data op een semantische wijze een toegevoegde waarde kan hebben voor het proces van valorisatie van data. Echter betekent het wel dat er data samengevoegd moet worden afkomstig uit verschillende bronnen. Verder is er voor het doel van “big data” analyse een behoefte aan systematische transformatie van aangeleverde data, zodat deze op een genormaliseerde en uniforme wijze opgeslagen en vervolgens onderzocht kan worden.

De directe aanleiding voor dit onderzoek is de behoefte aan een uniforme en gecentraliseerde wijze van data integratie - afkomstig uit verschillende bronnen, met wijzigende data en datastructuur - dat regelmatig uitgevoerd zal moeten worden.

1.2 Probleemstelling

Het lectoraat DIO/EMM doet een onderzoek naar de mogelijkheden tot een toepassing van een semantisch web als hulpmiddel bij kennisuitwisseling. Uit vorige onderzoeken is er gebleken, dat semantisch web een toegevoegde waarde zou kunnen leveren aan kennisuitwisseling. Er is door andere onderzoekers een proof of concept opgesteld. Op basis van deze proof of concept is er een aantal mogelijke problemen geconstateerd.

Voor het doel van kennisuitwisseling zijn er diverse databronnen beschikbaar gemaakt, zoals sensoren, databases en web-services. Het lectoraat wil graag weten of het mogelijk is om deze databronnen op een systematische wijze te integreren binnen een semantisch web. Het probleem ligt in de diverse natuur van deze databronnen en in het continu genereren van nieuwe data(soorten) door de onderliggende systemen.

Daarom is er een behoefte ontstaan aan een data integration framework, waarmee niet-semantische data zonder vaste structuur zodanig getransformeerd worden, dat deze in een semantische database (triplestore) opgenomen en geanalyseerd kunnen worden.

Het onderzoek sluit aan op de gekozen studierichting, software engineering, gezien het overeenkomstige kennisgebied. Eerst zal een onderzoek gedaan worden naar het semantische web en de mogelijkheden hiervan. De opgedane kennis wordt vervolgens toegepast bij een realisatie van een data integration framework.

1.2.1 Inhoudelijke keuzes

Om er voor te zorgen dat de onderzoek binnen de aangewezen tijd uitvoerbaar is en goed aansluit aan de situatie en behoefte binnen de organisatie, is er een aantal inhoudelijke keuzes opgesteld:

- Het scope van het onderzoek wordt beperkt tot integreren van data. Dat wil zeggen dat alle activiteiten ter doel van het verzamelen van data en het analyse van data buiten de scope van het onderzoek vallen.
- Het eindproduct van data integration is in een data formaat dat in een triple store geladen kan worden.
- Voor wat betreft beschikbare triple stores, wordt de keus beperkt tot Fuseki en Virtuoso
- Indien mogelijk, wordt er gebruik gemaakt van bestande open source oplossingen, ter ondersteuning van maintainability en extensibility.

1.2.2 Doelstelling

Het doel van het afstudeerproject is het onderzoeken van bestaande methoden en het uitbreiden hiervan ter ondersteuning van data integration processen op het gebied van het semantische web. Hiermee moet de onwenselijke situatie - waarbij de onderzoekers handmatige bewerkingen moeten uitvoeren, zonder een vaste methodiek – omgezet worden naar een wenselijke situatie, waarbij dit proces structuur krijgt en deels geautomatiseerd wordt.

1.2.2.1 Relevantie

De realisatie van dit framework is relevant voor het bedrijf zelf, maar ook voor de maatschappij en de groei van het kennisgebied. Deze relevantie wordt nader toegelicht.

1.2.2.1.1 Bedrijfsrelevantie

Het gerealiseerde data integration framework wordt allereerst toegepast op de Zeelandpascase. Hiervoor zijn er geanonimiseerde gegevens van het gebruik van de ZeelandPas vergeleken met andere beschikbare gegevens, zoals het weer en de evenementenkalender. De geleverde datasets zijn als verschillende bestandtypen geleverd. Deze datasets zijn niet genormaliseerd en de data is niet gestandaardiseerd (zo heeft bijvoorbeeld elke dataset een andere datumnotatie).

Op dit moment wordt de data handmatig geïntegreerd waardoor medewerkers van het lectoraat hier veel tijd aan moeten bestenden. Bovendien is dit proces niet gestandaardiseerd, waardoor elk medewerker een ander aanpak heeft. Combinatie van deze twee feiten zorgt voor een aanpak dat tijdrovend en foutgevoelig is.

De gerealiseerd framework moet hierop een oplossing bieden. Het geautomatiseerde data integratie volgens een gestandaardiseerde wijze, zou de organisatie veel manuren besparen en bovendien een data structuur dat beter onderhoudbaar is i.v.m. gestandaardiseerde wijze van werken.

1.2.2.1.2 Maatschappelijke relevantie

Het gerealiseerde framework zal het proces van data integratie standaardiseren, waardoor het makkelijker wordt om beschikbare bronnen op een semantische wijze op te slaan. Dat zou als effect kunnen hebben dat het aantrekkelijker wordt om bestaande, maar ook nieuw opgedane kennis volgens deze wijze op te slaan, met als uiteindelijke effect de groei van beschikbare kennis en het

gemak van het benaderen hiervan. Zo zouden bijvoorbeeld afstudeerrapportages beter bereikbaar zijn voor de studenten, wat de onderwijskwaliteit zal laten stijgen.

1.2.2.1.3 Kennisgebied en theoretische relevantie

Het kennisgebied van dit onderzoek kan als volgt beschreven worden:

- Informatica
    - Software Engineering
        - Data Integration
        - Semantic Web

Dit onderzoek vindt dus plaats binnen het kennisgebied van Informatica. Er wordt door middel van software engineering (dat een deelgebied is van informatica) naar oplossingen gezocht op het gebied van data integration, gebruikmakend van semantische web concepten (semantiek geven aan data, triple store).

Het semantische web is een vrij recent concept, dat gezien wordt als logische vervolgstap voor wat betreft de ontwikkeling van WWW (Hendler, 2009). Met het uitvoeren van het onderzoek en door realisatie van een data integration framework, kan er een bijdrage geleverd worden aan de groei van deze technologie.

Verder zal de realisatie van een data integration framework een bijdrage leveren aan het kennisgebied van data-analyse, doordat er een uniforme manier voor de normalisatie van data ontwikkeld wordt.

1.2.3 Vraagstelling

Op basis van de probleemstelling is er een hoofdvraag opgesteld, deze is vervolgens opgesplitst in deelvragen. Antwoorden op deze deelvragen, zullen gezamenlijk antwoord geven op de hoofdvraag.

1.2.3.1 Centrale vraag

Hoofdvraag luidt als volgt:

"Hoe kan een framework voor data integratie binnen de context van het lectoraat DIO worden gerealiseerd op basis van semantische web technologieën?"

1.2.3.2 Deelvragen

De hiervoor genoemde vraag valt uiteen in de volgende deelvragen:

- Wat is de huidige situatie van de techniek omtrent het semantische web en data integratie?
- Welke eisen stelt het lectoraat DIO aan een data integration framework?
- Hoe kan een data integration framework technisch en functioneel gerealiseerd worden, zodat deze aan alle eisen voldoet?

1.3 Theoretisch kader

De begrippen die gehanteerd worden bij dit onderzoek zullen in dit hoofdstuk nader gedefinieerd worden, zodat hierover geen verwarring kan ontstaan.

In dit hoofdstuk worden de belangrijkste begrippen benoemd en gespecificeerd. In de context van dit onderzoek, zijn de hier beschreven definities leidend.

1.3.1 Software framework

Het doel van het onderzoek is de realisatie van een framework. Maar wat is een framework eigenlijk?

"A software framework is a concrete or conceptual platform where common code with generic functionality can be selectively specialized or overridden by developers or users. Frameworks take the form of libraries, where a well-defined application program interface (API) is reusable anywhere within the software under development." (Technopedia, 2015).

Kortgezegd, een framework is een combinatie van libraries en op maat gemaakte software die als fundering voor verdere software development dient. Een framework bevat een gemeenschappelijk deel van een code, die door verschillende (bestaande, of nog te bouwen) modules gebruikt kan worden. Zo kan een framework bijvoorbeeld een code die verantwoordelijk is voor input en/of output, bevatten.

1.3.2 Data integration

Het te realiseren framework dient ter ondersteuning van het data-integratie proces. Volgens IBM staat data-integratie voor: *"Data integration is the combination of technical and business processes used to combine data from disparate sources into meaningful and valuable information"* (IBM, 2015). Onder data-integratie vallen alle processen ter ondersteuning van het samenvoegen van verschillende databronnen. Ook wordt hierin rekening gehouden met verschillende formaten en typen van data.

Dikwijls wordt er bij data integration gesproken over het proces van extraction, transformation en loading. Dit proces, ook wel **ETL** genoemd, wordt toegepast om verschillende datasoorten samen te voegen om zo hopelijk meer betekenis te geven aan data. Bij ETL wordt er vaak gebruik gemaakt van **Data Warehouse**, een gecentraliseerde verzameling van gegevens uit verschillende databronnen. Het gebruik van data warehouse zorgt ervoor dat er geen berekeningen uitgevoerd moeten worden op de originele databron. Deze hoeft immers niet altijd beschikbaar te zijn, en wordt ook frequent gebruik voor operationele processen van een bedrijf, waardoor performance vaak een issue is (Oracle, 2014).

1.3.3 Datanormalisatie

Normalisatie is het proces van organiseren van data in een database. Normalisatie wordt voornamelijk toegepast bij een relationele database. In een relationele database is het mogelijk om een relatie aan te geven tussen tabellen. Zo kan een tabel "bestelling" een relatie hebben met een tabel "klant" en "product". Het is voor een tabel bestelling belangrijk om een relatie met een klant en product te hebben, maar deze definiëren de tabel bestelling zelf niet (Microsoft, 2013).

Bij normalisatie wordt de data opgedeeld tussen tabellen, zodat elk tabel een logische "entiteit" beschrijft. Elk van deze entiteiten heeft een bepaalde relatie met een of meerdere andere entiteiten, die aanvullende gegevens kunnen bevatten. Een "bestelling" heeft een "klant"; "klant" daartegen heeft een "woonadres" en gegevens van "woonadres" kunnen gebruikt worden om de bestelling te verzenden.

Verder worden er bij normalisatie alle berekende waardes verwijderd. Data uit een database worden vaak gebruikt voor berekeningen, maar het is onmogelijk om van te voren te voorspellen wat er precies berekend zou moeten worden. Zo'n berekening kan altijd aangevraagd en worden uitgevoerd. Het resultaat hiervan hoeft niet opgeslagen te worden. Dat zorgt namelijk voor meer ruimte beslag, genereert meer acties die uitgevoerd moeten worden op een database en dat heeft weer een negatieve invloed heeft op de performance. Het kan in een worst-case-scenario deadlocks opleveren.

1.3.4 Semantic Web

Het thema van dit onderzoek is het semantische web. Het **Semantische web** is een vrij recent concept, waardoor het niet eenvoudig is om een duidelijke definitie te vinden. Zelfs het World Wide Web Consortium (W3C) reageert als volgt, wanneer gevraagd of er een definitie van semantisch web is.

"No formal definitions, but of course there are different approaches. Indeed, the complexity and variety of applications referring to the Semantic Web is increasing every day, which means that various application areas, implementers, developers, etc., would emphasize different aspects of Semantic Web technologies. This wide range of applications include data integration, knowledge representation and analysis, cataloguing services, improving search algorithms and methods, social networks, etc." (W3C, 2009)

Het is dus een vrij onduidelijk concept, dat verder geanalyseerd moet worden. **Semantic** heeft wel een duidelijke definitie: *"of, relating to, or arising from the different meanings of words or other symbols"* (Dictionary.com, 2015). Kortom, semantiek houdt zich bezig met de betekenis van een woord. Maar wat betekent dat in de context van het web?

Het huidige web wordt over het algemeen benaderd met browsers. Een gebruiker kan via een browser een HTML pagina bezoeken, waarin HTML tekst gezien kan worden. Deze tekst is voor een mens goed leesbaar, omdat wij van jongs af aangeleerd hebben wat de betekenis is van de woorden die we lezen. We weten ook dat er een bepaalde hiërarchie/categorisatie van deze woorden aanwezig is. Zo weten wij bijvoorbeeld dat roos een bloem is, dat bloemen planten zijn en dus is het logisch te beredeneren dat roos een plant is (Slaghuis, 2009).

Machines hebben dit begrip niet, waardoor veel betekenis van data verloren gaat. Er is geprobeerd om door middel van metadata meer betekenis te geven aan data, maar dat is oneindig veel werk. Per woord zouden er in metadata ALLE relevante relaties aangegeven moeten worden. In het eerder benoemde voorbeeld van een roos, zou je bijvoorbeeld in de metadata moeten specifiëren wat de kleur van een roos is, tot welke familie, orde, klasse enz. deze roos behoort, wanneer deze is geplant, wie de eigenaar is etc. Natuurlijk is deze aanpak niet realistisch en dus is er naar een andere oplossing gezocht.

Al sinds 1980 was men bezig om een oplossing te bieden aan dit probleem. Rondom 2000 is het begrip "semantic web" ontstaan, als beschrijving van een mogelijke oplossing (Semantic Web, 2010). Het idee achter semantic web is het geven van betekenis aan data, door het te koppelen met andere data. Een roos is van de stam "landplanten" en de stam "landplanten" is van het rijk "planten". Door middel van het semantische web zou het dus mogelijk moeten zijn om te constateren dat een roos een plant is.

Zo ontstaat er dus een set van "gelinkte" data. Veel mensen zijn bekend met een "boomstructuur". Een bekend voorbeeld hiervan is een stamboom. Elk element op een boom is gelinkt met een ander element of met andere elementen. Uiteindelijk is het ook mogelijk om op zo'n boom te gaan zoeken; er is immers een relatie tussen alle elementen, al is deze niet altijd direct.

Het semantische web maakt gebruik van de grafentheorie. Een graaf is vergelijkbaar met een boom; alleen is een graaf wat minder beperkt. Bij een boom is er sprake van een parent-child relation, een duidelijk begin- en eindpunt en duidelijk sprake van een richting, waardoor een cyclus geen optie is. Een graaf daarentegen kan relaties van verschillende types bevatten, bijvoorbeeld: "bevat"; "is onderdeel van"; "is eigenaar van" etc. Verder is er bij een graaf geen sprake van een duidelijk begin-

en/of eindpunt, noch van richting. Het is mogelijk om een graaf te maken die een cirkel is, of een graaf waar alle punten met elkaar verbonden zijn (Diestel, 2010).

Eigenschappen van grafen zorgen ervoor dat deze toegepast kunnen worden bij de semantic wiki. Zo kan elk element, met een willekeurig aantal andere elementen "gelinkt" worden en elk van deze links kan een betekenis hebben. Zo'n structuur zou het in theorie mogelijk moeten maken om extra betekenis aan data te geven, waardoor complexe query's mogelijk worden.

1.3.4.1 RDF

RDF staat voor Resource Description Framework en het wordt gebruikt voor het beschrijven van resources binnen World Wide Web. Met **resource** worden er alle "dingen" in de wereld bedoeld. *"In the Semantic Web we refer to the things in the world as resources; a resource can be anything that someone might want to talk about. Shakespeare, Stratford, 'The value of X' and 'all the cows in Texas' are all examples of things someone might talk about and that can be resources in the Semantic Web "* (Allemang & Hendler, 2011). De nogal vage definitie is te wijten aan de algemene aard van een resource, maar kort samengevat: een resource is iets wat relevant kan zijn en RDF is een framework voor het beschrijven van een resource.

Binnen RDF wordt gebruik gemaakt van **triples,** dat wil zeggen een 3-tuple. Een triple bevat een **subject**, een **predicate** en een **object**.

- Een subject omschrijft het te beschrijven object
- Een predicate beschrijf de relatie tussen een subject en een object
- Een object bevat de werkelijke waarde

Op deze manier is het mogelijk om een willekeurig aantal links tussen een willekeurig aantal elementen vast te leggen, waardoor er een "web of data" ontstaat. Wanneer een applicatie zich hiervan bewust is, kan er dus binnen dit web gezocht worden naar aanvullende informatie (Price, 2004).

1.3.5 Conflicttypen: Integratie van heterogene data

Bij integratie van heterogene data zijn er een aantal vaak voorkomende problemen te benoemen. Zo zijn er op dit gebied veel onderzoeken gedaan, waarbij een aantal vooronderstellingen en definities naar voren komen. Over het algemeen zijn er drie conflicttypen te benoemen die als oorzaak de heterogene karaktereigenschappen van data hebben (Gagnon, 2007):

- Syntactic heterogeneity
- Structural heterogeneity
- Semantic heterogeneity

Bij **Syntactic heterogeneity** worden alle conflicten veroorzaakt door verschillen in typen data modellen (entity-relationship, relational, object-oriented). Dit conflict kan opgelost worden met het type conversie. Echter, in een systeem waarin meerdere componenten verschillende formaten gebruiken, wordt het probleem groter. Normaal gesproken zou dat betekenen dat elk formaat "vertaald" moet kunnen worden naar elk ander formaat. Dat houdt in dat er *n(n-1)* conversiepaden zijn. Deze oplossing is zeer inefficiënt in de productie, maar ook in het onderhoud en dit probleem wordt vaak opgelost met een "transferformaat" die dit aantal reduceert tot *2n* (Worboys & Duckham, 2004).

Bij **Structural heterogeneity** worden alle conflicten veroorzaakt door verschillen in data structuur. Structural heterogeneity ontstaat wanneer datamodellen of de conceptualisering hiervan verschillen.

Een andere potentiële oorzaak is de representatie van hetzelfde concept op verschillende wijzen in verschillende databronnen (Embley & Thalheim, 2011).

Bij **Semantic heterogeneity** worden alle conflicten veroorzaakt door verschillende "betekenissen" van data. Data die op verschillende wijzen op een aantal plekken gedupliceerd zijn, kunnen dezelfde betekenis hebben (Hull, 1997). Dit soort conflicten zorgt ervoor dat potentieel veel betekenisvolle data verloren gaat, omdat de relaties tussen de entiteiten niet altijd aanwezig zijn.

1.3.6 Message translation

Bij de integratie van data moet er rekening gehouden worden met de bovengenoemde conflicttypen. Een mogelijke oplossing hierbij is de "message translation"-pattern. Deze pattern kan toegepast worden op één of meerdere niveaus:

- Data Structures
- Data Types
- Data Representation
- Transport

Voor het doel van dit onderzoek zijn twee niveaus relevant: Data Types en Representation.

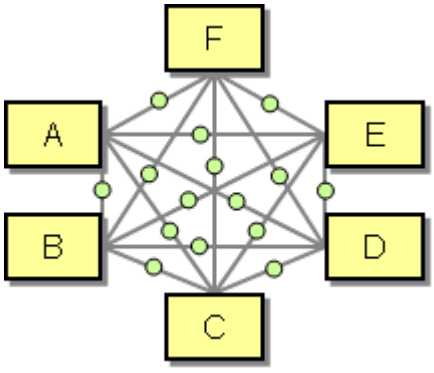
Data Types betreft: velden namen, data types (string, integer etc.), waarde domainen enz. kortom, deze eigenschappen kunnen problemen veroorzaken door eerder genoemd structural en semantic heterogeneity.

Data Representation betreft data formaten (zoals XML en CSV) en eventueel character-sets (bijvoorbeeld ASCII en Unicode).

Message translation kan toegepast worden op elk van deze niveaus, waarbij het sterk aan te raden is om een translation op elk niveau afzonderlijk te maken. Deze aanpak zorgt ervoor dat de code hergebruikt kan worden; een Data Representation translation is bijvoorbeeld niet afhankelijk van Data Type translation en deze hoeft dus niet voor elke message opnieuw gedefinieerd te worden (Hohpe & Woolf, Enterprise Integration Patterns, 2007).

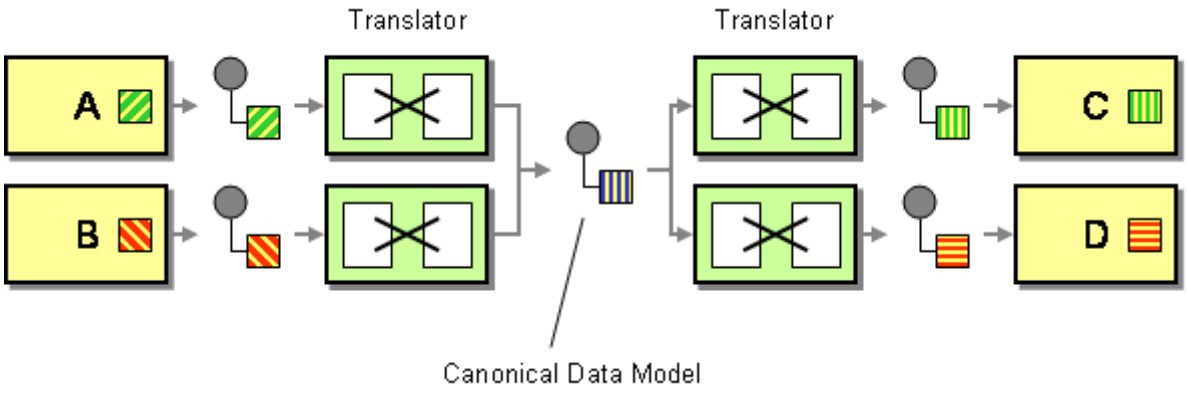
1.3.7 Canonical Data Model

Message translation kan een oplossing bieden bij problemen die voorkomen bij data integratie, maar deze heeft ook eigen beperkingen. Zoals bij Syntactic Heterogeneity genoemd, groeit het aantal translators exponentieel met de groei van applicaties die data uitwisselen (zie figuur 1). Dat levert veel problemen op. Ten eerste kost het veel tijd om te ontwikkelen, maar het zorgt vervolgens ook nog voor maintenance problemen. Wat moet er bijvoorbeeld gebeuren in een situatie waarbij een applicatie geüpdatet wordt, als er opeens een ander data-format gebruikt wordt?



Figuur 1

Een **Canonical Data Model** biedt hier een oplossing voor. In plaats van een translator voor elk conversie-pad, wordt er een tussenformaat gekozen. Alle data worden naar dit tussenformat vertaald, en vervolgens weer vertaald naar het benodigde format (zie figuur 2).



Figuur 2

Met canonical data model wordt de samenwerking tussen de applicaties wat meer indirect. Toevoegen van nieuwe applicaties wordt ook gemakkelijker; er zijn hooguit twee nieuwe translators nodig, ongeacht het aantal bestaande applicaties, dat is het voordeel van dit model.

De aanpak volgens het canonical data model zorgt echter voor een verhoogde overhead voor wat betreft de data-bewerkingen. De data moet per eenheid twee keer vertaald worden (naar en vanuit de Canonical Data Model) in plaats van een keer bij een directe vertaling (Hohpe & Woolf, Enterprise Integration Patterns, 2007).

1.3.8 Samenhang

Om de samenhang tussen alle in het theoretisch kader benoemde onderwerpen duidelijk te maken, is er een diagram opgesteld. In dit diagram zijn alle relevante begrippen (in dit document - paragrafen en dikgedrukte woorden) weergegeven evenals de bijbehorende relevante relaties. Dit diagram is in Bijlage 19 te vinden.

Kort samengevat: de opdracht is het maken van een software framework, met als doel data normalisatie en data integration. Deze bewerkingen moeten als effect RDF opleveren. RDF is een framework voor het beschrijven van resources voor het doel van semantic web. Deze resources worden als triples beschreven, waarbij elke triple uit een subject, object en predicate bestaat. Bij data integration wordt vaak ETL proces toegepast. Daarbij wordt data op een gestructureerde wijze in een data warehouse opgeslagen. Het is daarbij zeer belangrijk om op gestructureerde wijze te werken, omdat er bij data integration een aantal problemen kunnen voorkomen. Een belangrijk probleem daarbij zijn conflict typen. In dit onderzoek wordt er vanuit gegaan van drie conflict typen: semantic heterogeneity, structural heterogeneity en syntactic heterogeneity. Om de problemen bij data integration te kunnen oplossen, worden vaak “Message Translation” en “Canonical Data Model” patterns gebruikt.

2 Methode

In dit hoofdstuk wordt er per deelvraag aangegeven welke methoden gebruikt worden om een antwoord te kunnen bieden. Een gekozen methode wordt vervolgens nader toegelicht; zo worden het meetinstrument en de operationalisatie van de relevante parameters beschreven. Ook wordt de te gebruiken analysemethode uitgebreid omschreven. Alle in dit hoofdstuk genomen keuzes betreffende de methode, worden verder verantwoord.

2.1 Deelvraag 1: Wat is de huidige situatie van de techniek omtrent het semantische web en data integratie?

Om dit doel te realiseren is er een literatuuronderzoek uitgevoerd. Bij dit onderzoek is er gebruik gemaakt van de beschikbare wetenschappelijke literatuur en digitale bronnen.

2.1.1 Meetinstrument en operationalisatie

Resultaten uit het literatuuronderzoek zijn gebruikt als basis voor het vervolg van het onderzoek. Alle verkregen resultaten zijn in het zoekplan vastgelegd.

Verder waren er een aantal eisen gesteld, voordat de resultaten als “relevant” beschouwd konden worden:

- De te gebruiken bronnen worden indien mogelijk voorzien van de auteur.
    - Bij digitale bronnen zonder auteur, wordt de organisatie die verantwoordelijk is voor de bron als auteur benoemd.
- De te gebruiken bronnen worden indien mogelijk voorzien van de datum.
    - Bij het vermelden wordt de meest recente datum gebruikt (last modified).
    - Bij digitale bronnen zonder datum, wordt het meest actuele copyrightdatum benoemd.
- De te gebruiken bronnen worden indien mogelijk beperkt tot bronnen die in de laatste vijf jaar gepubliceerd zijn.
    - Als passende bronnen ontbreken in de publicaties van de afgelopen vijf jaar, wordt de zoekopdracht uitgebreid tot de laatste tien jaar.
    - Als goede bronnen desalniettemin ontbreken, vervalt het criterium van de datum.

Relevante resultaten betreffende de theorie van data integration, semantic web, software engineering zijn in het theoretisch kader beschreven. Alle relevante resultaten betreffende oplossingsrichtingen voor in de probleemstelling gestelde vraag, zijn in het hoofdstuk “Resultaten” onder deelvraag 1 beschreven.

2.1.1.1 Onderbouwing keuze meetinstrument

Het is aannemelijk dat er op het gebied van data integration al wetenschappelijke onderzoeken uitgevoerd zijn. Ook is de kans groot dat er software beschikbaar is ter ondersteuning van dit proces. Deze kennis, aangevuld met kennis van de situatie binnen het bedrijf, kan gunstig zijn bij het oplossen van het probleem van de hoofdvraag.

Een literatuuronderzoek naar de huidige situatie van de techniek kan een beeld geven van wat er al op het gebied van data integratie en het semantische web bekend is. Zo kunnen er strategieën, belangrijke dilemma’s en software pakketten die het proces ondersteunen naar boven gebracht worden, en geanalyseerd worden voor wat betreft de toepasbaarheid binnen dit onderzoek.

De redenatie achter in de vorige paragraaf genoemd aanpak is als volgt: het onderzoek moet traceerbaar en leesbaar zijn door iedereen. Elke vraag moet beantwoord kunnen worden met een zo recent en betrouwbaar mogelijke bron; maar bij gebreke hiervan, kunnen de eisen versoepeld worden, om toch een antwoord te kunnen bieden.

2.1.2 Analysemethode

Bij de analyse is gebruik gemaakt van de eerder genoemde eisenlijst, gecombineerd met de analyse van de betrouwbaarheid van data. Betrouwbaarheid wordt bepaald op basis van het gebruikte medium. Zo is een wetenschappelijk artikel of boek als meer betrouwbaar dan een privé blog of forum. De analyse beschrijft de huidige stand van zaken van het semantische web en data integratie dient. Deze analyse is vervolgens als basis voor het vervolgonderzoek gebruikt.

De gevonden oplossingsrichtingen zijn geanalyseerd op relevante voor- en nadelen. Om dit proces leesbaar te maken, werd er per oplossing een tabel gemaakt met voor- en nadelen en opmerkingen.

2.2 Deelvraag 2: Welke eisen stelt het lectoraat DIO aan een data integration framework?

Om een goed beeld van de eisen en verwachtingen te krijgen, die aan de software gesteld worden, is er een requirementsanalyse uitgevoerd. Voor dit doel zijn er een aantal methoden beschikbaar, zoals enquêtes, focus groups en interviews. Binnen dit onderzoek is er voor gekozen om het interview te kiezen.

Met het onderwerp “Data integration” zijn er binnen DIO voornamelijk twee personen bezig: Anton Bil en Hans de Bruin. Beiden hebben een beeld van het probleem, maar dit verschilt nogal in het perspectief. Zo is Anton Bil een ervaren gebruiker, die streeft naar “power user” functies en het automatiseren van tijdrovende bewerkingen. Hans de Bruin is meer gericht op de toepasbaarheid in de toekomst binnen een groter systeem. Er is dus een sprake van twee stakeholders die een duidelijk, maar toch wel verschillend beeld hebben van het probleem.

2.2.1 Meetinstrument en operationalisatie

Er is een aantal interviews met boven genoemde stakeholders gehouden. De bevindingen van deze interviews zijn volgens een MoSCoW methode vastgelegd en als tussenproduct ingediend.

Dit proces is als volgt verlopen:

- De onderzoeker stelt een vragenlijst op;
- De onderzoeker maakt een afspraak met een desbetreffende stakeholder;
- Het interview begint met open vragen over het algemene beeld van de te realiseren software;
- Er wordt gedetailleerd doorgevraagd op basis van de antwoorden op eerder gestelde open vragen;
- Alle vragen van de vragenlijst komen aan bod (eventueel gesloten vragen);
- De requirements worden vastgelegd volgens de MoSCoW-methode;
- Het resultaat wordt gepresenteerd aan de geïnterviewde.

Op basis van voortschrijdend inzicht is deze procedure verder uitgebreid. Zo is er besloten om alle interviews op te nemen en om te zetten naar een transcript.

Vragenlijsten en MoSCoW-notaties worden als bewijsstukken geleverd en zijn in de bijlagen te vinden (zie Bijlage 1 en Bijlage 2).

Ook is er besloten om de scope in kaart te brengen. Op basis van de interviews is het bedrijfsproces vastgelegd in een model (zie Bijlage 3), aangevuld met de aanduiding welke elementen van dit proces aangepast kunnen worden.

Op basis van het scopemodel, en een aantal brainstorm sessies met de opdrachtgever, is er bovendien een "Problem Mapping" opgesteld. Het Problem Mapping bestaat uit twee elementen: grafische weergave van de potentiële problemen per stap van een proces, en de verdere specificatie in tekstuele vorm per potentieel probleem. Deze is in Bijlage 4 en Bijlage 5 te vinden.

2.2.1.1 Onderbouwing keuze meetinstrument

De redenering achter de methodekeuze is dat een interview een beter beeld zal geven van probleem, omdat er meer tijd besteed wordt aan de verschillende interpretaties van het probleem. Overige methoden zijn meer geschikt bij grote groepen stakeholders en zijn over het algemeen minder nauwkeurig omdat de resultaten vaak geaggregeerd moeten worden.

De verkregen resultaten zijn volgens de MoSCoW-methode genoteerd; deze methode is gedurende de opleiding gehanteerd, waardoor er veel ervaring is opgebouwd met deze methode. Ook zorgt deze notatie er voor dat er in een ogenblik requirements zichtbaar gemaakt worden, zodat het niet meer nodig is om alle informatie door te nemen die via de interviews verzameld is.

De scope is in kaart gebracht omdat de eerder opgestelde probleemstelling onvoldoende was. Het was belangrijk om goed beeld te hebben wat de bedrijfsprocessen zijn, en welke daarvan aangepast kunnen worden.

Met deze scope, was het mogelijk om potententiele problemen te benoemen per stap van het bedrijfsproces. Deze aanpak zorgt er voor dat niet alleen de potentiële problemen bekend zijn, maar ook op welk moment ze kunnen voorkomen. Met deze informatie is het mogelijk om een aangepast bedrijfsproces te ontwerpen, dat van deze problemen vrij is.

2.2.2 Analysemethode

De prioriteit van de requirements wordt als volgt opgesteld:

- Must – requirements waaraan het eindproduct moet voldoen om als "voltooid" beschouwd te worden.
- Should – requirements die zeer belangrijk zijn, maar waar het ontbreken ervan niet zal leiden tot het "onbruikbaar" stellen van het product
- Could – in het geval er nog tijd beschikbaar is, zijn dat de requirements die overwogen kunnen worden ter implementatie
- Won't – requirements die niet opgenomen worden in deze iteratie van het eindproduct.

Waar mogelijk, worden requirements gekoppeld aan het antwoord waarvan het requirement is afgeleid, door middel van een requirement id (REQ_XXX).

De opgestelde lijst van requirements is vervolgens gebruikt om een ontwerp te maken, en het gerealiseerd framework te valideren.

De validiteit van scope en problem mapping is vastgesteld door de opdrachtgever.

2.3 Deelvraag 3: Hoe kan een data integration framework technisch en functioneel gerealiseerd worden, zodat deze aan alle eisen voldoet?

Om dit doel te kunnen realiseren is er eerst onderzocht welke ontwerpen het meest geschikt zijn voor dit project. Naarmate de onderzoeker meer kennis en ervaring kreeg met de eisen en de

omgeving, is het duidelijk geworden welke ontwerptechnieken (modellen/diagrammen) van toepassing zijn.

Het opstellen en volgen van een ontwerp bevordert de kwaliteit van software en het volgen van best practices. Verder zorgt een ontwerp ervoor dat architectuur van een gerealiseerd product gemakkelijker geanalyseerd kan worden. Een duidelijke ontwerp zorgt ervoor dat software onderzocht kan worden door een software engineer, met als doel maintenance van software.

2.3.1 Meetinstrument en operationalisatie

Als resultaat van deze deelvraag zijn er twee documenten opgeleverd: het Functioneel Ontwerp en het Technisch Ontwerp. Elk van deze documenten is een samenvoeging van de bijbehorende modellen en diagrammen, aangevuld met de specifieke context van het project.

Het functioneel ontwerp en de bijbehorende modellen (Use cases en high-level sequence diagram) geven een beeld van de te realiseren functionaliteit en de samenhang tussen verschillende functionaliteiten. Het functioneel ontwerp kan gebruikt worden om tot een consensus te komen met de opdrachtgever over de te realiseren functionaliteit, voordat deze nog werkelijk gerealiseerd wordt.

Het technisch ontwerp en de bijbehorende modellen (deployment diagram en datamodel) zijn een schematische weergave van de werking en de gebruikte technieken binnen de te realiseren software. Het technisch ontwerp is dan ook voor een deel een blauwdruk, waarop de software gebaseerd is. Dit ontwerp werd gebruikt als leidraad tijdens het maken van software, maar het kan ook in latere stadia van de software cyclus gebruikt worden, ter ondersteuning van het maintenance proces.

Het ontwerp is vervolgens op een iteratieve wijze gerealiseerd. Hierbij werd gebruik gemaakt van PoC (Proof of Concept) en Prototypes.

Het PoC werd gebruikt om te bewijzen dat de gekozen techniek een oplossing zou kunnen bieden om het probleem op te lossen. Bij een PoC wordt er een versimpeld probleem voorgesteld – bijvoorbeeld een mapping tussen niet relevante data – om aan te tonen dat de software de functionaliteit kan realiseren. Het voordeel hiervan is dat er geen rekening gehouden hoeft te worden met de context van data, waardoor het maken en testen sneller kan verlopen.

Prototypes daartegen, zijn toegepast op de echte scenario's, bij dit onderzoek was dat de ZeelandPas case. Bij een prototype werd het echte situatie nagebootst, om te controleren of het gerealiseerde framework ook in de echte situatie van toepassing blijft.

PoC en prototypes zijn vervolgens geanalyseerd op het gebied van de requirements en de problem mapping. Deze zijn gepresenteerd aan de relevante stakeholder en door hen voorzien van feedback.

2.3.1.1 Onderbouwing keuze meetinstrument

Het maken van ontwerpen bevordert de kwaliteit van software, aangezien het peer review proces hierdoor sneller kan verlopen. Het is ook een extra moment waarbij feedback gegeven kan worden door de relevante stakeholders. Bovendien zijn ontwerpen het fundament voor de ontwikkelaar, wat vooral relevant is in geval van langdurige projecten. Uiteindelijk kan een ontwerp als documentatie middel gebruikt worden, ook door toekomstige ontwikkelaars.

Voor dit onderzoek is er gekozen om gebruik te maken van een iteratieve ontwikkelingsmethode, waarbij er PoC's en prototypes gebouwd worden. Dat heeft een aantal voordelen. Allereerst betreft dit project een kennisgebied dat voor de onderzoeker nog relatief onbekend is, waardoor er een gebrek aan expertise aanwezig is. Door PoC's en prototypes te presenteren aan de relevante

stakeholders - waaronder wat meer ervaren software engineers – is het mogelijk om het project sneller bij te sturen. Ook levert dat weer extra feedback momenten op die het leerproces bevorderen. Een aantal korte software development cycli zorgt er bovendien voor dat nieuw verkregen kennis en inzicht sneller toegepast kan worden. Bovendien wordt het proces van bug tracking en bug fixing simpeler gemaakt, omdat het makkelijker wordt om erachter te komen welk aanpassing in de code tot problemen heeft geleid.

Succesvolle realisatie van het prototype zal de technische correctheid van de oplossing bewijzen. Het verschil tussen de ZeelandPas case, en andere potentiële cases ligt in de gebruikte databestanden en de grootte daarvan en de gebruikte mapping. De mapping is generiek van aard, en kan dus zonder problemen afwijken. Wat wel onderzocht moet worden is hoe het prototype omgaat met zeer grote bestanden.

2.3.2 Analysemethode

Functionele requirements worden gecontroleerd door middel van user acceptance test. De onderzoeker zal eerst zelf vaststellen of er aan de requirements voldaan wordt. Vervolgens wordt deze aan de gebruiker gepresenteerd, samen met het desbetreffende requirement.

Om het technische gedeelte van het framework te testen, zijn er twee testen uitgevoerd.

Allereerst is er een performance test uitgevoerd. Het testen van “ondersteuning van bestanden groter dan 500MB” eis was relevant, maar ook hoe lang zo’n proces kan duren.

Voor dit doel is er een aantal datasets gebruikt. Op het internet is er naar grote .csv bestanden gezocht, en zo is er .csv-bestand van ~2,94GB gevonden (Lemire, 2012). Als dit bestand de toepassing blokkeerde, dan is een test met een kleiner dataset nodig. In dit geval, wordt de eerder gebruikte dataset verder in kleinere datasets gesplitst: ~972MB en ~486MB. Daarbij staat 972MB gelijk aan bijna het dubbele van de gestelde eis, en 486MB zit er net onder.

Aangezien het framework uit twee elementen bestaat, zijn deze ook afzonderlijk getest. Bij de OpenRefine waren er 3 controle momenten: load, save en export. Deze drie representeren de stappen van OpenRefine tijdens het opschonen van data. Eerst worden de data ingeladen, waarbij er een preview gegenereerd wordt. Vervolgens worden deze data opgeslagen in een project. Zodra dat gebeurd is, kunnen er wijzigingen in data aangebracht worden met het doel data op te schonen. De resulterende dataset kan vervolgens geëxporteerd worden. Van elk van deze tests wordt de tijdsduur gemeten, de gebruikte max_memory instellingen, de gebruikte machine en de maximale heap usage.

Bij de RML-Processor verliep de test als volgt. Eerst worden de gegevens van input genoteerd: aantal ingeladen bestanden, en de size hiervan. Vervolgens wordt het tijdstip bij de start van de test genoteerd. Ook aan het einde van de test wordt het tijdstip genoteerd, en wordt de tijdsduur berekend. Ook wordt de size van de output genoteerd.

Vervolgens is er een accuracy test uitgevoerd. Om de correctheid van de data te controleren zijn er acht JUnit tests aangemaakt en uitgevoerd. Er is besloten om in eerste instantie gebruik te maken van simpele synthetische tests. Zo zijn de volgende elementen getest: datum, (transaction) Id, pasnummer, pastype, pasuitgever, pasuitgever postcode, postcode en VVXH. Om de tests zo automatisch en volledig mogelijk te laten verlopen, is er een JUnit test suite opgebouwd, waarin de eerder genoemde elementen getest konden worden.

Deze tests zijn binnen het JUnit framework opgebouwd(zie Bijlage 15). Het idee was om de tests zo volledig en automatisch mogelijk te laten verlopen; daarom is er voor mechanische test gekozen. De reden dat deze tests zijn in Java gerealiseerd omdat de gebruikte triple store een Java API had.

Binnen elk van deze tests zijn er 2 SPARQL query's uitgevoerd. Een query op de compareset en een query op de gegenereerd set. De resultaten van deze query's zijn vervolgens per lijn met elkaar vergeleken. Deze aanpak zorgt er voor, dat het aantal triples gelijk moet zijn evenals elke triple afzonderlijk.

Vervolgens zijn de transacties vergeleken. Eerst is er een poging gedaan om deze test met de originele mapping uit te voeren. Deze mapping bleek echter niet equivalent, en dus is er een nieuwe mapping aangemaakt.

3 Resultaten

Op basis van in de vorige hoofdstuk genoemd methoden, zijn er resultaten verkregen. Deze worden in dit hoofdstuk nader toegelicht. Dit hoofdstuk is opgesplitst in twee onderdelen: Resultaten en Analyse.

Bij het onderdeel resultaten worden de verkregen resultaten zo objectief mogelijk beschreven. Dat is belangrijk, omdat deze resultaten vervolgens geanalyseerd moeten worden. In geval dat er al interpretaties plaats zouden vinden, zal dat betekenen dat de analyse op al "bewerkte" gegevens gebaseerd zou worden – "lost in translation" effect.

Bij het onderdeel analyse worden de eerder verkregen resultaten geïnterpreteerd en geanalyseerd. Daarbij wordt er een poging gemaakt om interessante en/of onverwachte resultaten uit te leggen. Voor het doel van analyse wordt er gebruik gemaakt van in de methode hoofdstuk beschreven analyse methode.

3.1 Deelvraag 1: Wat is de huidige situatie van de techniek omtrent het semantische web en data integratie?

In deze paragraaf worden de resultaten van het literatuuronderzoek beschreven. Zoals in het hoofdstuk "methode" is genoemd, worden hier de mogelijkheden toegelicht om het probleem zoals in de probleemstelling verwoord, op te lossen. De theorie omtrent software frameworks, data integration en semantische web is in het theoretisch kader te vinden.

3.1.1 Resultaten

RDF-bestanden kunnen gebruikt worden om data in een triple store in te laden. Echter, de aangeleverde bestanden zijn meestal van andere typen, zoals CSV of XLSX. Om de daar opgeslagen data toch in een triple store te kunnen laden, kunnen deze *geconverteerd* worden. Dat wil zeggen: van een bestand type naar een ander bestand type omzetten.

Uit het onderzoek is het gebleken dat er een lijst bestaat van converters, die verschillende bestandstypes omzetten naar een RDF (W3C, 2014). Een van de voor CSV beschikbare converters is Google Refine. Uit de in de Wiki beschikbare beschrijving blijkt deze ook Excelsheets en andere vormen van tabellarische data opslag te ondersteunen. Dat is natuurlijk zeer interessant voor het doel van dit onderzoek, omdat het framework zoveel mogelijk formaten moet ondersteunen.

3.1.1.1 OpenRefine

OpenRefine (voormalig Google Refine) is een tool voor het werken met "vieze rommelige" data. Deze tool maakt het mogelijk om data op te schonen en te transformeren. Dat sluit aan bij het doel van het framework, omdat de te integreren data vaak eerst opgeschoond moet worden (OpenRefine, 2015).

De bruikbaarheid van OpenRefine voor het doel van het onderzoek wordt bepaald door twee criteria: ondersteunde bestandstypes en automatisatie.

3.1.1.1.1 Ondersteunde bestandstypes

Ten eerste is het zeer belangrijk om te onderzoeken of OpenRefine RDF ondersteunt. Gezien het feit dat OpenRefine gevonden werd op een lijst van RDF converters (W3C, 2014), lijkt dat vanzelfsprekend. Toch, voor het doel van volledigheid moet het onderzocht worden. Bovendien konden er wijzigingen in de software plaats vinden, die verder niet meegenomen werden in de lijst van converters. Dit is iets dat onderzocht moet worden.

Zo is het duidelijk gebleken dat RDF niet standaard ondersteund wordt en is hier een extensie voor nodig; deze breidt de oorspronkelijke software uit met RDF-opties. Deze extensie heet RDF Refine en is open source beschikbaar (Digital Enterprise Research Institute, n.d.). Hiermee worden RDF-bestanden dus ondersteund.

Volgens het boek “Using OpenRefine” worden o.a. volgende bestandstypes ondersteund:

- CSV, TSV en andere *SV
- XLS en XLSX
- ODF en ODS
- JSON
- XML
- Line-based formats

Verder wordt er opgemerkt dat niet ondersteunde typen kunnen ondersteund worden doormiddel van extensies (zoals RDF).

3.1.1.1.2 Templates opslaan

Een groot probleem bij het gebruik van OpenRefine is het handmatige kopiëren van templates. In de huidige situatie moeten er per bestand vier invoervelden ingevuld worden om een template te genereren. Deze kunnen vervolgens niet binnen OpenRefine zelf opgeslagen worden. Als workaround worden deze templates er op dit moment binnen media wiki bijgehouden. Wel betekent het dat er elk keer, per bestand, vier velden gekopieerd moeten worden.

Tijdens het onderzoek is er ontdekt dat het mogelijk is om een RDF-skeleton aan te maken. Deze kan vervolgens in een vorm van JSON opgeslagen of ingeladen worden. Deze RDF-skeleton bevat RDF types van cells. Wanneer deze types gedefinieerd zijn, is het mogelijk om een RDF/XML te exporteren (Digital Enterprise Research Institute, n.d.).Ook is het mogelijk om elk overige bewerking als JSON template te extrageren, en in de volgende projecten toe te passen.

Deze ontdekking is een vooruitgang op de huidige situatie, maar belangrijker nog – deze zorgt ervoor dat templates opgeslagen kunnen worden. De aanname is dan ook, dat deze templates vervolgens gebruikt kunnen worden, om via de code diezelfde export te genereren, wat een nog grotere verbetering zou zijn ten opzichte van de huidige situatie.

3.1.1.2 Karma

Een verder onderzoek naar beschikbare data integration tools heeft Karma opgeleverd. Karma is een tool voor de integratie van data in een triplestore. Verder maakt Karma het mogelijk om data te modelleren door gebruik te maken van een grafische interface (Information Sciences Institute, University of Southern California, 2014).

Het proces van modelleren wordt deels geautomatiseerd. Karma analyseert eerder uitgevoerde acties en maakt voorstellen op basis van de gebruikte ontologie, beperkingen zoals domeinen en ranges. Verder is het mogelijk om data te transformeren, al lijkt deze functionaliteit meer beperkt dan die van OpenRefine. Het resultaat van deze bewerkingen kan een bestand zijn, maar er is ook een optie om de data direct in het triplestore in te laden. Het kan dus overwogen worden om OpenRefine voor transformaties en conversies te gebruiken, en Karma om data te modelleren en in een triplestore te laden.

3.1.1.3 LDIF

LDIF staat voor Linked Data Integration Framework. Deze tool biedt de mogelijkheid om data te verzamelen door gebruik te maken van SPARQL, Crawling of file download (RDF/XML, N-triples, N-quads en turtle). Deze data worden vervolgens gemodelleerd, waarbij de identiteit van data wordt vastgelegd. Het resultaat van deze acties is een N-Quad of N-Triple file output of een bijbehorende store (University of Mannheim, 2014).

Qua functionaliteit lijkt dit framework op Karma, al lijkt LDIF zich meer richten op quadstores en is functionaliteit voor triple stores pas later gekomen. Verder moet opgemerkt worden dat LDIF minder goed onderhouden lijkt te zijn; op moment van schrijven was de laatste update op 13/2/2014 uitgevoerd. Verder is de laatste reactie op de Google group van LDIF op 1/8/2014 geplaatst, met de vraag of de software nog ondersteund wordt. Op deze vraag is er geen reactie gegeven. LDIF kan dus als optie overwogen worden, maar er moet rekening gehouden worden de vraag of deze software nu al, of in de nabije toekomst ondersteund kan worden.

3.1.1.4 R2RML

Tijdens het onderzoek naar integratie van heterogene data, is er een paper ontdekt dat RML als oplossing biedt. RML is een extensie op R2RML en daarom is er besloten om eerst de basis te analyseren.

R2RML is een taal voor het mappen van relationele databases naar een triple store. Deze mapping wordt als een RDF-graaf opgeslagen en kan dus met gemak geïntegreerd worden in een triple store. R2RML maakt het mogelijk om een relationele database als een triple store te benaderen, gebruikmakend van RDF vocabulaire en datastructuur die door de ontwikkelaar van mapping bepaald zijn. R2RML gebruikt een relationele database als input en levert een RDF dataset (SPARQL) als output. Deze is echter conceptueel en kan omgezet worden naar een “echte dataset” of behouden worden als virtuele toegang tot de aanwezige informatie (W3C, 2012).

R2RML is een recommandatie van W3C ter ondersteuning van data integratie met een relationele database.

3.1.1.5 RML

RML is een uitbreiding op R2RML en is ontwikkeld om een oplossing te bieden voor de limitaties van R2RML, waarbij R2RML het mogelijk maakt om relationele databases te integreren. RML breidt deze functionaliteit uit naar CSV, JSON en Xml-bestanden. Dit doel wordt bereikt doordat de referenties naar relationele database specifieke eigenschappen meer generiek gemaakt worden. Ook wordt de iterator expliciet benoemd, zodat de wijze van doorlopen van een datacontainer expliciet gespecificeerd wordt (Dimou, et al., 2014).

Het is belangrijk om op te merken dat RML maar een “public working draft” is. Dat wil zeggen dat het geen ondersteuning heeft van een officiële organisatie en het kan aangepast worden.

3.1.1.6 SKOS

SKOS is een model voor het delen en verbinden van data in het web. SKOS is ontwikkeld op basis van KOS – knowledge organization systems – die gebruikt werden voor het organiseren van grote collecties van voorwerpen, zoals museumartefacten of boeken.

Al snel is ontdekt dat vergelijkbare een systeem gebruikt kan worden ter ondersteuning van het organiseren van beschikbare data. Vooral voor wat betreft ongestructureerde data, kan SKOS een positieve bijdrage leveren aan de classificatie en categorisatie van data.

SKOS is ontwikkeld om KOS in een machine-leesbare taal weer te geven. Deze kan vervolgens gebruikt worden om uitgewisseld te worden met andere machines, bijvoorbeeld via het web of via applicaties.

SKOS classificeert de organisatie van kennis als “concept scheme”, bestaande uit meerdere “concepts”. Een concept bevat een URI, waardoor er hiernaar ondubbelzinnig gerefereerd kan worden. Elk concept kan vervolgens een aantal elementen bevatten, zoals:

- labels (in verschillende talen, met verschillende alternatieve benamingen)
- notaties (zoals symbolen)
- documentatie (bijvoorbeeld definities)
- links (relaties met andere concepten)

SKOS is een W3C recommandatie (W3C, 2009), waardoor deze betrouwbaarder is voor het gebruik.

3.1.2 Analyse

Binnen dit sub-paragraaf worden de gevonden technieken geanalyseerd op hun voor- en nadelen.

3.1.2.1 OpenRefine

Voordelen	Nadelen	Opmerkingen
Ondersteund CSV, TSV, XLS, XLSX, ODF, ODS, JSON, XML en line-based formats	RDF ondersteuning niet native (extensie)	Het open source natuur van de software, verhoogd extensibility
Open source	“Per-source” conversie. Koppelingen tussen verschillende bestanden kunnen niet vastgelegd worden.	Lijkt niet goed om te gaan met zeer grote data bestanden, iets om onderzocht te worden.
Templates (RDF skeleton) kunnen opgeslagen worden voor hergebruik	Lijkt meer bedoeld voor handmatige bewerking dan automatisatie (gezien het feit dat er voor elk bestand een nieuwe project gemaakt moet worden, i.p.v. data als verwisselbaar invoer element)	Templates worden opgeslagen als JSON geformatteerd tekst. Het is niet mogelijk om deze als onderdeel van een project o.i.d. op te slaan in een client. Uiteindelijk betekent het dat de template ergens als tekst opgeslagen moet worden, en bij hergebruik, als tekst gekopieerd.
Tegelijkertijd een tool voor opschonen van data		OpenRefine is niet gebouwd met RDF in gedachte (extensie). Kan impact hebben op maintainability.

3.1.2.2 Karma

Voordelen	Nadelen	Opmerkingen
Modelleren geautomatiseerd op basis van eerdere bewerkingen	Transformatie kant minder uitgebreid	Karma is gebouwd met semantische data opslag in gedachte. De verwachting is dat deze elementen daardoor beter ondersteund worden.
Maakt gebruik van aanwezige ontologie	Stabiliteit lijkt een issue te zijn. Tijdens het onderzoek is het	

	software aantal keren vastgelopen. Bovendien reageert deze slecht op zaken zoals undo/redo, go back (vermoedelijk i.v.m. browser-based natuur van software)	
Kan data direct in een triple store inladen		

3.1.2.3 LDIF

Voordelen	Nadelen	Opmerkingen
Import via crawlers	Ondersteund geen data bestanden (CSV, JSON etc.)	LDIF lijkt niet meer ondersteund te worden.
Hoge mate van automatisatie (Scheduler)		Lijkt meer gericht op data dat al semantisch opgeslagen is, en dus – niet relevant.
Kan data direct in een triple store inladen		

3.1.2.4 R2RML

Voordelen	Nadelen	Opmerkingen
Automatische conversie op basis van een template (mapping)	Ondersteund alleen maar relationele databases/views/SQL query's als input	Basis voor RML
Template in de vorm van een turtle notatie		Een W3C recommandatie
Het resultaat kan in een triplestore ingeladen worden.		Het is een taal en geen implementatie op zichzelf.

3.1.2.5 RML

Voordelen	Nadelen	Opmerkingen
Automatische conversie op basis van een template (mapping)	Geen officiële implementaties.	Gebaseerd op R2RML
Ondersteund relationele databases, maar ook CSV, TSV, JSON, XML		Unofficial draft
Ondersteund meerdere bestanden tegelijkertijd		Het is een taal en geen implementatie op zichzelf

3.1.2.6 SKOS

Voordelen	Nadelen	Opmerkingen
Levert extra beschrijving aan de opgeslagen data	Meer data dat opgeslagen moet worden (performance)	Het is een data model ter ondersteuning van ordenen van kennis
Er bestaan al veel succesvolle implementaties.		

3.2 Deelvraag 2: Welke eisen stelt het lectoraat DIO aan een data integration framework?

In deze paragraaf zijn de resultaten van de uitgevoerde requirementsanalyse te vinden. Eerst worden de gebruikte termen nader toegelicht. Vervolgens wordt er per interview de vragenlijst en notities beschreven.

Op basis van deze antwoorden zijn er requirements opgesteld. De opgestelde requirements zijn dus een interpretatie van de eisen en verwachtingen van de stakeholders.

3.2.1 Resultaten

Om de leesbaarheid van het document te vergroten wordt deze sub-paragraaf verdeeld in de volgende onderwerpen: Toelichting gebruikte termen en Interview.

3.2.1.1 Toelichting gebruikte termen

In het hoofdstuk met de "resultaten" is er een aantal nieuwe termen aan bod gekomen. Om de leesbaarheid en de nauwkeurigheid van het document te bewaken, worden deze hier toegelicht.

Template – met template wordt binnen dit onderzoek een deel van een code bedoeld, dat ingevoerd wordt binnen een framework, om het specifieke deel van de functionaliteit van de code te realiseren. Een template is hier dan ook een aanvulling op het generieke gedeelte van framework, zodat bijzondere situaties die niet generiek te beschouwen zijn, toch uitgevoerd kunnen worden. Zo kan een template bijvoorbeeld beschrijven hoe een tekst veld verdeeld moet worden in afzonderlijke velden.

Data bestanden – met data bestanden worden alle bestanden bedoeld die gebruikt worden voor gestructureerde opslag van data. Een voorbeeld hiervan is CSV (Comma Separated Values) of XLS (Microsoft Excel Spreadsheet).

Unattended – met unattended wordt bedoeld dat de applicatie gedurende een proces geen input vraagt aan de gebruiker. In dit geval is de gebruikersinput van tevoren geleverd, in tegenstelling tot **Attended** waarin er continue met de gebruiker wordt geïnteracteerd (door middel van input schermen, pop-ups en dergelijke).

3.2.1.2 Interview

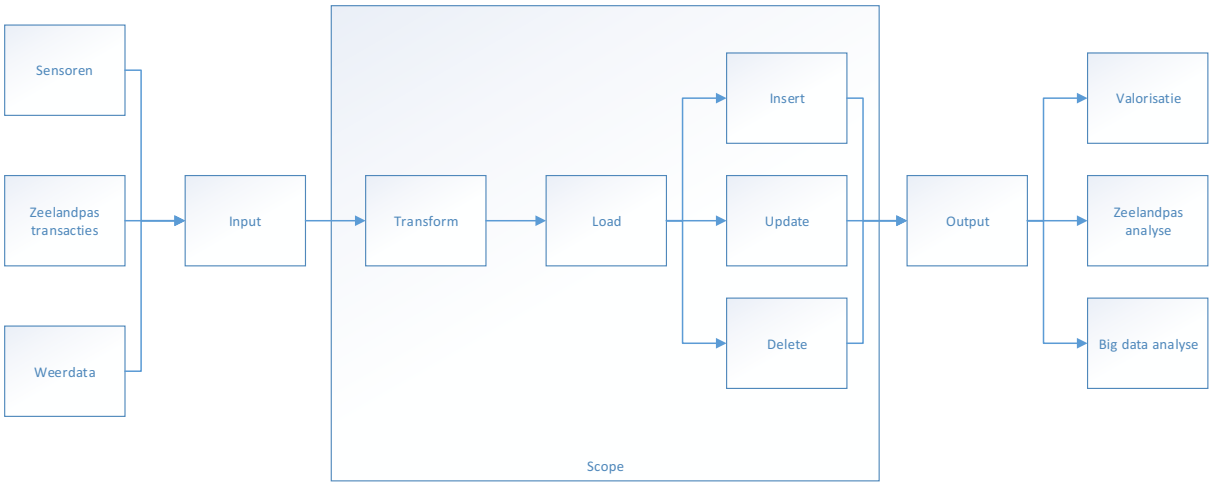
De resultaten van het interview zijn opgenomen in de bijlagen van dit rapport (zie Bijlage 1). Het interview met Anton Bil is uitgewerkt op basis van snel gemaakte notities tijdens het interview. Het interview met Hans de Bruin is uitgewerkt met behulp van een geluidsopname, en het heeft dus de vorm van een transcript.

3.2.2 Analyse

In deze sub-paragraaf worden de resultaten van het interview geanalyseerd. Zo wordt het scope bepaald, vervolgens worden op basis van deze scope potentiële problemen benoemd. Uiteindelijk is er een MoSCoW-lijst opgesteld met de requirements.

3.2.2.1 Scope

Tijdens het onderzoek is het duidelijk geworden dan het zeer sterk aan te raden is om de scope van het project goed te bepalen. Omdat het te realiseren framework een onderdeel moet zijn van een langer proces, moet het duidelijk zijn welk stappen dit framework moet ondersteunen. Voor dit doel is er een diagram opgesteld.



Figuur 3

In figuur 4 is te zien dat er duidelijk sprake is van een langer proces, waarbij het framework geen invloed heeft op de verkregen input, en geen stappen onderneemt nadat er een output gegenereerd is. De bijbehorende acties/processen (zoals het verzamelen van data en het analyseren hiervan) vallen dus buiten de scope van het project. Dat houdt dus ook in dat - voor het doel van onderzoek – de leverancier van data als externe partij beschouwd kan worden. Samenwerking met externe partijen heeft als effect dat de mogelijkheden om data te beïnvloeden beperkt zijn. Het is mogelijk om afspraken met hen te treffen, maar er moet ook rekening gehouden worden met het feit dat deze niet altijd volledig naar wens uitgevoerd kunnen worden.

Binnen de scope worden de volgende stappen genomen:

- Transform – alle acties/processen die ter verbetering van kwaliteit van data dienen
- Load – alle acties/processen die het mogelijk maken om data in een gewenst format (triple store) in te laden
- Insert/update/delete – alle acties/processen die tot doel hebben de data in een triple store te manipuleren

Het te realiseren framework dient een oplossing te bieden voor de problemen die kunnen voortkomen uit het in de scope genoemde proces.

3.2.2.2 Mapping van het probleem

Op basis van de opgestelde scope was het mogelijk om potentiële voorkomende problemen te mappen. Dat zijn de problemen die uit de gesprekken met de stakeholders en in de theorie naar voren zijn gekomen. De opgestelde mapping is in de bijlagen te vinden (zie Bijlage 4 en Bijlage 5).

3.2.2.3 MoSCoW

Om de resultaten op een overzichtelijke manier te presenteren is er een requirements-lijst opgesteld volgens de MoSCoW notatie. Deze is in de bijlagen te vinden (zie Bijlage 2).

3.3 Deelvraag 3: Hoe kan een data integration framework technisch en functioneel gerealiseerd worden, zodat deze aan alle eisen voldoet?

In deze paragraaf worden FO & TO toegelicht in een afzonderlijke sub-paragraaf. Vervolgens wordt er op basis van het opgestelde prototype onderzocht of er aan de eerder opgestelde eisen voldaan wordt.

3.3.1 Resultaten

Er is gekozen om gebruik te maken van OpenRefine en RML. Dit gebeurt op basis van oplossingsrichtingen die voor de deelvraag 1 gevonden zijn en de opgestelde eisen in deelvraag 2.

Allereerst werd overwogen om alleen OpenRefine te gebruiken. Dat komt omdat OpenRefine een groot deel van de gewenste functionaliteit zelf kan realiseren. Echter, naar aanleiding van het interview met Hans de Bruin, is het duidelijk geworden dat OpenRefine op een aantal punten te kort schiet. Zo houdt OpenRefine geen rekening met een aantal potentiële problemen, zoals het samenvoegen van meerdere bestanden tegelijkertijd. OpenRefine is een voorbeeld van een "per-source" aanpak van het data integration probleem, waarbij er een bestand ingeladen wordt, om dit vervolgens te converteren en in te laden. Wanneer er meerdere bestanden geconverteerd moeten worden, gebeurt dat in een serie, zonder dat er verbindingen tussen deze bronnen vastgelegd worden.

Vanwege de beperkingen van OpenRefine zijn ook andere opties te overwegen. Vooral R2RML leek veelbelovend; het betreft een aanpak die niet meer per-source is. Het is namelijk mogelijk om een aantal bestanden tegelijkertijd te integreren (met alle bijbehorende relaties). Echter heeft deze oplossing een belangrijk nadeel: het is een per-format aanpak. Dat wil zeggen dat er weliswaar met meerderde bestanden gewerkt kan worden, maar deze moeten wel van hetzelfde type zijn, namelijk een relationele database. Veel van de data die beschikbaar zijn op het internet wordt opgeslagen in een relationele database; echter in het geval van lectoraat DIO wordt er veel van de data ontvangen van externe leveranciers. Het overdragen van dit soort data gebeurt eerder via exports, dan met directe toegang tot de relationele database. Het is wel mogelijk om dit soort exports vervolgens in een relationele database te laden. Dat zou er voor zorgen dat er een extra tussenstap gelegd moet worden en daarom is er gezocht naar alternatieven.

RML is een uitbreiding op R2RML; dat wil zeggen dat alle functionaliteiten van R2RML ook via RML uitgevoerd kunnen worden. Maar, terwijl R2RML zich puur op relationele databases richt, kan RML ook op andere gebieden gebruikt worden (zoals CSV, JSON en XML). Er moet rekening mee gehouden worden met het feit dat RML redelijk recent en experimenteel is. In het geval dat RML niet goed (genoeg) blijkt te werken, kan er overwogen worden om een stap terug te doen naar R2RML.

Overige alternatieven schieten te kort op bepaalde gebieden. Zo was Karma redelijk onstabiel, wat voor veel frustratie bij de gebruikers zou kunnen zorgen. LDIF daarentegen ondersteunt geen databestanden en dat is juist het grote aandeel van de geleverde data.

Het uiteindelijke product is dus een combinatie van OpenRefine en RML-processor. Om een stap-voor-stap handleiding van deze toepassing te bekijken, zie Bijlage 18.

3.3.1.1 Functioneel Ontwerp

Om een globaal beeld van de gewenste functionaliteit vast te leggen, zijn er een aantal use cases opgesteld. Deze use cases zijn opgesteld op basis van de in de vorige deelvraag vastgelegde eisen naar concrete functionaliteit. Om een beeld te krijgen van alle use cases zie use case diagram in Bijlage 6.

Binnen deze use cases kan er gesproken worden over twee typen actors: expert en niet-expert. Tijdens de interviews is er een beeld ontstaan dat er in de werkelijke situatie sprake is van twee soorten gebruikers: een expert met een ICT-achtergrond en een niet-expert, die weliswaar een beetje verstand heeft van computers (opstarten van applicaties, verplaatsen van bestanden), maar van wie geen deskundige ICT kennis verwacht kan worden.

De opgestelde use cases kunnen in de bijlagen gevonden worden (zie Bijlage 6).

Op basis van deze use cases is er een high-level sequence diagram opgesteld. Een sequence diagram wordt meestal als een technisch ontwerp beschouwd, maar hier is er voor gekozen om dit op high level te houden. De reden hiervoor is dat er een sprake is van een duidelijke, stapsgewijze (sequence) aanpak, en dat er daardoor behoefte is aan een weergave van dit proces.

Dit highlevel sequence diagram kan in de bijlagen gevonden worden (zie Bijlage 7).

Bij dit diagram is er sprake van een actor (user), die alle in de use case genoemde activiteiten wil uitvoeren. Ook zijn er een aantal alternatieve paths te benoemen. Zo moet er eentje aangemaakt worden in het geval dat er nog geen project aanwezig is.

Verder kan de actor bepalen of het project meteen als RDF/XML gegenereerd kan worden en daardoor RML Processor kan overslaan. Dat is niet aan te raden, aangezien er een wens is voor een gestructureerde werkwijze, maar het is wel iets dat in bepaalde situaties overwogen kan worden.

3.3.1.2 Technisch Ontwerp

Aangezien er gebruik gemaakt wordt van bestaande oplossingen is er voor gekozen om gebruik te maken van een deployment diagram, om duidelijk aan te wijzen op welke locatie deze toepassingen gehost moeten worden. Dat bleek eenvoudiger te zijn dan verwacht; alle elementen kunnen namelijk het beste op een server gehost worden.

OpenRefine is voorzien van een web interface, waardoor het mogelijk is om deze ook van buitenaf te benaderen. Verder stelt OpenRefine hoge eisen aan het RAM-geheugen. Over het algemeen is het te verwachten dat een server meer RAM-geheugen bezit dan een desktop of een laptop.

RML Processor kan in geval van complexe transformaties veel tijd nodig hebben om deze uit te voeren. Het is sterk aan te raden om deze transformaties in een batchmodus, bijvoorbeeld in de nachturen, te laten uitvoeren.

Beide elementen kunnen natuurlijk ook local gehost worden; echter heeft dat een aantal nadelen. Ten eerste zorgt het ervoor dat er minder gestructureerd gewerkt wordt. Zo kan elk gebruiker een ander mappenstructuur hebben, waardoor de RML mappings aangepast moeten worden. Ook is het natuurlijk de bedoeling dat de geconverteerde bestanden uiteindelijk in een triple store geladen worden, en deze bevindt zich op een server.

Er zijn dus redenen om de recommandatie voor al deze elementen (voor wat betreft productie omgeving) op een server te houden, met een van te voren bepaalde mappenstructuur. Een testomgeving kan in dit geval natuurlijk wel local gehouden worden. Het gerealiseerde ontwerp is in de bijlagen te vinden (zie Bijlage 8).

Voor het doel van het prototype is er een datamodel opgesteld. Het is belangrijk om op te merken dat deze niet gebaseerd is op een standaard. De reden hiervoor is dat het onmogelijk bleek om een goede tool te vinden voor het representeren van RDF op een grafische wijze. Tijdens het realiseren van het prototype was de volgende informatie nodig:

- Welke key attributes zijn er (op basis hiervan wordt URI opgebouwd)?
- Welke attributen horen bij elke key?
- Waar liggen de relaties tussen key attributen?

Tijdens het onderzoek is er gebruik gemaakt van een aantal verschillende tools; deze bleken echter onstabiel en te omslachtig. Daarom is er voor gekozen om een simpel diagram te maken met nodes en verbindingen ertussen.

Elk centrale node staat hier voor een key attribute (deze kan gezien worden als een klasse, of een primary key van een relationele tabel). Verder zijn er binnen het diagram een aantal nodes waarbij er tweerichtings connectie is aangelegd – deze zijn gebruikt om afzonderlijke waardes met elkaar te combineren. Binnen RML is het een simpele join condition, waarbij de twee waardes met elkaar moeten overeenkomen.

Het diagram zelf is gerealiseerd met VUE 3.2.2 en is in de bijlagen te vinden (zie Bijlage 9).

De RML-processor wordt aangeleverd met een class diagram - gemaakt door de ontwikkelaars van RML-processor - en hoewel dit tijdens het onderzoek zelf niet gebruikt werd, kan dit wel een positieve bijdrage leveren aan de maintainability van deze toepassing. Dit class diagram is in de bijlagen te vinden (zie Bijlage 11).

3.3.1.3 Proof of Concept

Om er voor te zorgen dat de te gebruiken software aan de eisen van de software kan voldoen, zijn er eerst een aantal “Proof of Concepts” opgebouwd. Elk van deze PoC is een bewijs van een bepaalde functionaliteit die de software mogelijk maakt. Er zijn de volgende PoC’s gebouwd:

1. Kleinschalige mapping met 1 bron
2. Kleinschalige mapping met een join binnen 1 bron
3. Kleinschalige mapping met meerdere bronnen
4. Grootschalige mapping met 1 bron

Elke PoC had als doel het testen van een bepaalde functionaliteit. Deze PoC’s worden nader toegelicht.

PoC 1 had als doel het bewijzen dat de software überhaupt werkt zoals bedoeld. Daarin is er een zeer simpele mapping gemaakt, binnen 1 bron, waarbij er maar een paar attributen zijn vertaald naar een turtle notatie (.ttl).

PoC 2 had als doel het bewijzen dat het mogelijk is om een mapping te maken, waarbij er een join gebruikt wordt. Hier is er voor gekozen om het eerst binnen 1 bron te testen, om het bouwen van mapping gemakkelijker te maken.

PoC 3 had als doel het bewijzen dat het mogelijk is om binnen 1 mapping meerdere bronnen te hebben. Daarbij is er bovendien nog gebruik gemaakt van een aantal joins.

PoC 4 was voornamelijk bedoeld om te kijken hoe de RML Processor omgaat met zeer grote datasets. Daarbij is gebruik gemaakt van een dataset van ongeveer 2,94GB.

Uiteindelijk zijn alle 4 PoC’s succesvol gebouwd. Wat wel op te merken is, is het feit dat er weinig feedback vanuit de RML Processor gegeven werd, in het geval dat de mapping niet klopte. In het geval dat er prefixen gebruikt werden die niet in de mapping stonden, of wanneer de bestand-pad niet correct was, werd er in de console een foutmelding gegeven met de foutbeschrijving. Echter, in alle overige gevallen werden er alleen maar Java debug informatie weergegeven. Dat heeft op een

aantal momenten ervoor gezorgd, dat de mapping niet werkte en dat het niet duidelijk was waarom. Soms ging het om een ontbrekende punt, andere keer weer om een typefout.

De mappings zelf zijn in de bijlagen te vinden onder PoC (zie Bijlage 12).

3.3.1.4 Prototype

Om te controleren of de gekozen oplossing van toepassing is in de werkelijkheid en de context van het bedrijf, is er een prototype gebouwd. Dit prototype moet het ZeelandPas Case nabootsen. Daarbij is er een datamodel opgebouwd (zie Bijlage 9), gebaseerd op eerder gebruikte handmatige templates. Dit model is vervolgens gepresenteerd aan de relevante stakeholders. Op basis van daaruit voortvloeiende feedback, is het datamodel uitgebreid. Een aantal elementen die relevant waren voor het onderzoek ontbraken en moesten nog toegevoegd worden. Vervolgens is dit datamodel weer gepresenteerd, en goedgekeurd.

Het datamodel is vervolgens gerealiseerd tot een mapping (zie Bijlage 13). Deze mapping werd vervolgens gebruikt om de geleverde testdata te vertalen naar .ttl . De verkregen .ttl is vervolgens in de triple store (Fuseki) zonder foutmeldingen ingeladen.

Dit prototype is vervolgens gebruikt om een aantal testen en metingen uit te voeren.

3.3.1.4.1 Performance Metingen

Voor het doel van performance metingen, zijn er twee machines gebruikt. Deze hebben de volgende specificatie:

PC	
Processor	Intel Core i7-4770K
Processor clock	Vaste, 3,5GHz
OS	Windows 8.1 Pro x64-bit
RAM	16 GB
Storage	Samsung 840 EVO 500GB
Read speed	540 MB/s
Write speed	520 MB/s

Laptop	
Processor	Intel Core i7-3632QM
Processor clock	Vaste, 2,2 GHz
OS	Windows 8.1 Pro x64-bit
RAM	8 GB
Storage	Hitachi Travelstar Z5K500
Read speed	100 MB/s
Write speed	38 MS/s

Bij elke meting wordt er aangegeven welke machine er gebruikt werd.

OpenRefine performance

Een van de opgestelde requirements was het inladen van grote (tot 500MB) bestanden. De verwachting is dat het RAM-geheugen hierbij een belangrijke rol zal spellen.

Ten eerste is er begonnen met een CSV-bestand van 2,94GB. Het doel was het inladen van het bestand (load), het opslaan van een project (save) en vervolgens exporteren hiervan in een CSV-vorm (export)

Machine	Laptop
File Size	2,94GB
Max memory	4096MB
Heap usage	3817/3817MB
Load Time	00:02:15:3
Save Time	Fail (00:28:42:0)
Export Time	Fail

Machine	Laptop
File Size	2,94GB
Max memory	5120MB
Heap usage	4772/4772MB
Load Time	00:02:28:0
Save Time	Fail (00:17:32:7)
Export Time	Fail

Tijdens het meten is er geconstateerd dat de CPU en het memory gebruik 100% hebben bereikt. Bij een poging tot het opslaan van het project, was het gebruik 100%, totdat de in de savetime genoemde tijdsduur, is bereikt. Daarna is het gebruik gedaald, en leek de applicatie niet meer responsief. De OpenRefine server rapporteerde een latency van ~1000ms.

Gezien de teleurstellende resultaten is er voor gekozen om gebruik te maken van een sterkere machine. Dit heeft de volgende resultaten opgeleverd.

Machine	PC
File Size	2,94GB
Max memory	10240MB
Heap usage	9544/9544MB
Load Time	00:00:29:5
Save Time	Fail (00:19:16:6)
Export Time	Fail

Machine	PC
File Size	972MB
Max memory	4096MB
Heap usage	3817/381MB
Load Time	00:00:29:7
Save Time	Fail (00:07:54:5)
Export Time	Fail

Bij een tweede meting is er voor gekozen om het CSV-bestand in kleinere stukken te splitsen. Bij een grootte van 972MB is het nog steeds niet gelukt om het project op te slaan. Vervolgens is er besloten om een bestand te creëren, dat aan de requirements voldoet. Hiervoor is er een CSV-bestand gemaakt met de grootte van 486MB.

Machine	Laptop
File Size	486 MB
Max memory	4096MB
Heap usage	3808/3817MB
Load Time	00:00:59:6
Save Time	Fail (00:30:00:0)
Export Time	Fail

Machine	Laptop
File Size	486 MB
Max memory	5120MB
Heap usage	4769/4772MB
Load Time	00:00:40:5
Save Time	00:01:39:0
Export Time	00:03:22:7

Zoals het hier te zien is, was het met 4GB ram (4096MB) nog steeds niet mogelijk om het project op te slaan. Gezien de tijdbeperking is er na 30 minuten besloten, om de meting te stoppen. Dit was de eerste keer dat het software het “crash-symptoom” niet vertoonde, dat bij de vorige metingen wel zichtbaar was (daling van gebruik van CPU en Memory). Het verhogen van het beschikbare geheugen tot 5GB (5120MB) heeft het wel mogelijk gemaakt om het project op te slaan, en vervolgens te exporteren.

RML Processor

Allereerst is er begonnen met een CSV-bestand van 2,94GB. Op basis van dit bestand is er een mapping gebouwd, met daarin 2 attributes, en 1 key. Vervolgens is er gebruik gemaakt van de ZeelandPas case mapping (voor specificatie zie datamodel van ZeelandPas case). De totale grootte van hiervoor de gebruikte bestanden is ongeveer 4,04MB. De resultaten waren als volgt:

	Machine	Laptop
Input	Size	2,94GB
	Amount	1
Time	Start	12:24:00
	End	15:34:20
	Duration	03:10:20
Output	Size	1,23GB

	Machine	Laptop
Input	Size	4,04MB
	Amount	4
Time	Start	15:26:00
	End	15:32:21
	Duration	00:06:21
Output	Size	48,8MB

3.3.2 Analyse

Binnen deze sub-paragraaf worden de resultaten van het ontwerp en prototype geanalyseerd. Zo worden eerst de performancemetingen geanalyseerd. Vervolgens wordt de correctheid van de geconverteerde data geanalyseerd.

3.3.2.1 Performance

OpenRefine

Een meting die als opmerkelijk gezien kan worden, is het feit, de applicatie na een bepaalde tijdsduur “het leek op te geven”. Dit was het geval bij grote databestanden en kleine hoeveelheid RAM. Gedurende het proces werd een groot deel van de processorkracht en het RAM geheugen in gebruik genomen, en op den duur daalde het verbruik sterk. Dit feit werd niet genoteerd in de server console, noch werd er een foutmelding weergegeven in het UI. Het feit dat de applicatie het niet deed, is dan weer niet zo opmerkelijk, maar het is te verwachten dat er een foutmelding of iets dergelijks weergegeven zou worden.

Ook is het opmerkelijk dat het RAM-verbruik veel hoger is dan de grootte van het bestand. Bij een bestand van 486 MB, was er 4779 MB RAM nodig; dat is bijna 10 keer zo veel. Het is natuurlijk wel te verwachten dat er meer RAM nodig zou zijn dan het bestand zelf; er moet immers eerst het originele bestand ingeladen worden, het nieuwe bestand moet er aangemaakt worden, en in de tussentijd wordt er een aantal transformaties uitgevoerd. Toch lijkt de factor 10 niet echt efficiënt.

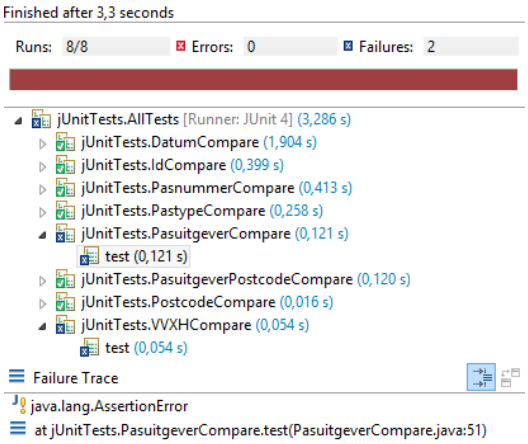
Uiteindelijk was het eis 500 MB en dat is wel gelukt met 5GB RAM. Een gemiddelde laptop heeft al gauw 4 GB, en bij de ICT professionals kan er 8 GB verwacht worden. Bovendien is de kans groot dat de applicatie op een server deployed zou worden, en servers beschikken over het algemeen over grotere hoeveelheden RAM geheugen.

RML Processor

Het is gelukt om een zeer groot CSV-bestand in te laden en te laten transformeren. Uit de resultaten lijkt het alsof deze processor niet RAM gebonden is. Dus bij deze component kan er gezegd worden dat de grootte van het ingeladen bestand een non-issue is. Naarmate het bestand, en de complexiteit van de mapping groeit, wordt het resultaat echter groter; ook duurt de transformatie zelf langer.

3.3.2.2 *Correctheid van data*

De resultaten van de tests zijn in figuur 5 te zien. Van de acht uitgevoerde tests, zijn er zes geslaagd en twee mislukt. Dat betekent dat bij de zes geteste result-sets de volgorde en inhoud precies hetzelfde zijn. De verwachting was wel dat alle acht tests zouden slagen; daarom is er verder onderzoek gedaan naar de oorzaak.



Figuur 4

Om achter de oorzaak te komen zijn de twee testen visueel getest. Daaruit bleek wat de oorzaak was. In het geval van PasuitgeverCompare, lag het aan de notatie van de letter “é”. In geval van VVXH, lag het aan de notatie van de values. VVXH bevat waardes vanaf 1 t/m 24, kortom: getallen/integers. In de dataset die ter vergelijking gebruikt werd, zijn deze values ook als integer genoteerd, RML processor heeft echter deze omgezet naar Strings, en dus ook tussen aanhalingstekens gezet. Bovendien worden Strings op een ander wijze gesorteerd: 11 wordt als kleiner gezien dan 2, omdat deze als tekst geïnterpreteerd wordt.

Vervolgens werden er transacties vergeleken, voor de gebruikte tests zie Bijlage 15. Al snel werd het duidelijk dat de datasets niet direct vergelijkbaar zijn. Allereerst is er het issue van verkeerd vertaalde tekens en data types. Maar een belangrijker issue hierbij is de verschillende datastructuur. In de opgestelde mapping werd er in meerdere gevallen uitgegaan van een volledige join; dat wil zeggen een instantie waarbij in beide bronnen dezelfde informatie beschikbaar was. Echter, in meerdere gevallen bleek het anders te zijn. Zo waren veel van de plaatsnamen niet aanwezig in gemeentes.csv. Dat zorgde er weer voor dat het aantal triples anders was dan in de originele dataset, waardoor het vergelijken hiervan niet meer mogelijk was.

```
Total: 3758.0
Matching: 3607.0
Not Matching: 151.0
Percent not matching: % 4.0180945
```

Figuur 5

Daarom is er besloten om een nieuwe mapping aan te maken, met alleen de data die nodig is voor vergelijking (zie Bijlage 14 en Bijlage 15). De data van deze nieuwe mapping heeft dus ook dezelfde structuur. Nu verliep de test veel soepeler. Zoals in figuur 6 te zien is, zijn er uit 3758 triples maar 151 die anders waren dan de vergelijking-set. Voor de test zelf was de foutmarge op 5% gezet, dus deze test kan als succesvol beschouwd worden. De vraag is, waarom zijn niet alle triples hetzelfde? Het antwoord is zeer simpel: plaatsnamen. Zo staat er in de ene data set “pashouder_plaats = "Sch”neck"” en in een ander “?pashouder_plaats = "Sch?neck"”. Kortom, het gaat om een al eerder geconstateerd probleem met Nederlandse tekens.

4 Discussie

In dit hoofdstuk worden er conclusies getrokken op basis van de verkregen resultaten. Allereerst wordt er per deelvraag gediscussieerd over de validiteit van het uitgevoerde onderzoek en de verkregen resultaten. Ook wordt er een deelconclusie getrokken. Vervolgens wordt er een eindconclusie getrokken, en wordt het gelopen onderzoektraject vergeleken met andere onderzoeken/theorie. Ook worden er suggesties gegeven voor wat betreft de mogelijke vervolgonderzoeken. Uiteindelijk wordt er recommandatie gegeven voor wat betreft de implementatie van het ontwikkelde systeem.

4.1 Deelvraag 1: Wat is de huidige situatie van de techniek omtrent het semantische web en data integratie?

In deze paragraaf wordt de discussie beschreven over de deelvraag: “Wat is de huidige situatie van techniek omtrent het semantische web en data integratie?”. Hieruit volgt een (deel)conclusie.

4.1.1 Discussie

De gebruikte methodologie heeft zeer veel resultaten opgeleverd. Een grote deel hiervan was ook van wetenschappelijke aard. Er kan dus aangenomen worden dat dit redelijk betrouwbaar is. Wel moet men zich afvragen, of het niet verstandiger zou zijn om eerst de wat meer “globale theorie” te bestuderen, in plaats van de technische. In dit onderzoek zijn eerst de technische boeken over RDF, SPARQL en design patterns bestudeerd. Deze waren natuurlijk zeer leerzaam, maar de gevonden oplossing heeft er uiteindelijk weinig mee te maken. Ja, de toegepaste techniek maakt gebruik van design patterns. En ja, verstand van SPARQL en RDF was nodig bij het analyseren hiervan. De vraag is echter of het niet gemakkelijker zou zijn om gerichter de techniek te bestuderen op basis van de gevonden oplossingen?

4.1.2 Deelconclusie

Op basis van het uitgevoerde onderzoek, kan er geconcludeerd worden dat RDF en het semantische web nog in een beginstadium zijn, vooral op het gebied van data integratie. Er worden pogingen gedaan om voor deze kwestie een oplossing te bieden; echter is de aanpak hiervan vrij chaotisch en gedecentraliseerd. Zo zijn er veel teams die met zeer verschillende wijzen van werken, een oplossing proberen te bieden. Ook zijn de meeste oplossingen in een vroeg stadium van development, en veel daarvan zijn nog eens voortijdig gestrand. Wel is het duidelijk dat deze oplossingen vorm beginnen te krijgen. Het lijkt nog te vroeg om te zeggen dat deze echt stabiel en klaar zijn om in een productie-omgeving neer te zetten. OpenRefine lijkt daarbij een goede zet in de richting van opensource-data-opschonen-tools. Doordat deze opensource en uitbreidbaar is, heeft deze een potentie om snel en efficiënt te groeien. Ook kan deze vervolgens “in-house” aangepast worden aan de behoeften van een organisatie. Voor wat betreft transformatie van data, lijkt de RML-Processor een stap in de goede richting. Doordat deze vrij generiek is op gebied van iterators en references, lijkt het dat deze nog verder uitgebreid kan worden naar andere formaten.

4.2 Deelvraag 2: Welke eisen stelt het lectoraat DIO aan een data integration framework?

In deze paragraaf wordt de discussie beschreven over de deelvraag: “Welke eisen stelt het lectoraat DIO aan een data integration framework?”. Hieruit volgt een (deel)conclusie.

4.2.1 Discussie

Het onderzoek naar deelvraag 2 is niet probleemloos verlopen. De gekozen methode lijkt nog steeds het beste van toepassing vanwege het kleine aantal respondenten en het belang voor de

verschillende interpretaties van het probleem. Wel was het nodig om tijdens het onderzoek een aantal verbeteringen aan te brengen, zoals het opnemen van interviews.

Vermoedelijk lag het probleem vooral aan de onduidelijke probleemstelling. Nadat deze verduidelijkt was, verliep het onderzoek beter, vooral omdat er gerichter vragen gesteld konden worden.

Een andere potentiële verbetering zouden vaste afspraken voor wat betreft reviews kunnen zijn. Vaak waren er lange perioden waarin weinig gecommuniceerd werd. In deze tussenliggende perioden divergeerden de visies van de opdrachtgever en de onderzoeker.

Uiteindelijk is er besloten om de ZeelandPas Case als basis te gebruiken, en de verkregen resultaten te analyseren op potentiële uitbreiding. De aanname hierbij is dat er op deze manier een redelijk accuraat antwoord gegeven kan worden op het “grotere probleem” binnen het bedrijf.

4.2.2 Deelconclusie

Uit de requirements analyse blijkt dat het lectoraat DIO geïnteresseerd is in een framework om op systematische wijze om te gaan met de integratie van data. Hierbij betreft het een zoveel mogelijk geautomatiseerd en gestandaardiseerd proces dat om kan gaan met grote datasets (zie REQ_002 en REQ_007). Dit proces moet gemakkelijk herhaalbaar zijn, door het proces van modelleren eenmalig uit te voeren (zie REQ_006, REQ_009, REQ_016). De data die geïntegreerd moeten worden zijn niet-semantisch van aard en komen van een derde partij (zie REQ_001, REQ_009, REQ_013). Wel kunnen er afspraken gemaakt worden voor wat betreft de structuur hiervan. Het resultaat moet een vorm krijgen van semantische data, die vrij is van dubbele en niet-genormaliseerde data.

Ook is het uit de scope en problem mapping duidelijk geworden dat de oplossing een verbetering moet zijn op al bestaande bedrijfsproces, dat vergelijkbaar is met een ETL proces. Daarin zijn meerdere problemen geconstateerd, zoals duplicatie van data, en niet actuele data, dat opgelost moeten worden met de data integration framework.

4.3 Deelvraag 3: Hoe kan een data integration framework technisch en functioneel gerealiseerd worden, zodat deze aan alle eisen voldoet?

In deze paragraaf wordt de discussie beschreven over de deelvraag: “Hoe kan een data integration framework technisch en functioneel gerealiseerd worden, zodat deze aan alle eisen voldoet?”. Hieruit volgt een (deel)conclusie.

4.3.1 Discussie

Op basis van de uitgevoerde tests lijkt het erop dat er wel degelijk goede resultaten zijn behaald, maar in de loop van het onderzoek wordt duidelijk dat de methode waarschijnlijk beter kan. Zo is er in eerste instantie niet goed nagedacht over de wijze van testen van het gerealiseerde framework. Het idee van vergelijken van oude en nieuwe data is pas later ontstaan, met als effect dat het gemaakte prototype en de zo ontstane dataset niet equivalent waren met de vergelijkingsset. Daardoor was het nodig om een nieuwe mapping aan te maken, zodat de test toch uitgevoerd kon worden. De verwachting is, dat het proces sneller en efficiënter zou verlopen wanneer er hiermee rekening gehouden zou worden. Dat is belangrijk, omdat meer beschikbare tijd omgezet zou kunnen worden naar extra functionaliteit en betere kwaliteit tests.

De validiteit van de test, vooral de TransactieCompare kan in twijfel gebracht worden, aangezien er een geheel nieuwe mapping is aangemaakt, wat in het begin niet de bedoeling was. Ook is er een kwestie van ~4% van de data die blijkbaar niet geheel correct is.

Voor wat betreft de mapping zelf: de bedoeling van de laatste test was om te controleren of de dataconversie correct is uitgevoerd. Het vergelijken met de oude dataset controleert twee zaken: dataconversie en datastructuur. Met behulp van de nieuwe dataset is er alleen maar op correctheid van dataconversie gecontroleerd. Uiteindelijk was dat ook het doel van de test, dus de validiteit van de test hoeft niet in twijfel gebracht te worden voor wat betreft dit punt. De eerste mapping was niet geschikt voor deze test vanwege de aannames die gemaakt zijn bij het maken van de desbetreffende mapping. Daarin is van meerdere JOINS uitgegaan - zoals het feit dat elk stad en elk dorp in transactielijst ook aanwezig is in de lijst met gemeentes - wat als effect had dat de data niet overeenkwam.

Voor wat betreft de ~4% van de data die niet overeenkomt, is de aanname dat deze door Nederlandse tekens komt die aanwezig waren in beide bronnen. De vergelijkingsset is met behulp van een andere applicatie vertaald naar semantische data, waardoor deze Nederlandse tekens ook op andere wijzen zijn geconverteerd. In beide gevallen was deze vertaling incorrect - blijkbaar kunnen beide tools niet omgaan met de Nederlandse tekens – maar deze fout heeft zich op een afwijkende wijze gemanifesteerd. Bijvoorbeeld, in het geval van “Café”, heeft RML processor deze als “Cafâ€š” vertaald, en in de vergelijkingsset stond deze als “Cafï¿½”. Kortom, beide zijn incorrect, maar doordat ze afwijkend zijn, kunnen ze niet vergeleken worden. Het probleem komt bij de oude aanpak ook voor, dus er worden in principe geen nieuwe problemen gegenereerd. De vraag is dus in hoeverre deze situatie een probleem is. Het is aannemelijk dat deze situatie problematisch kan zijn in twee gevallen: bij de weergave van data en bij het zoeken naar data.

Met weergave wordt bedoeld dat de naam als een label gebruikt kan worden. Echter wordt het als best practices beschouwd om voor dit doel een apart attribuut aan te maken, namelijk een label. Ook is het zoeken naar data via naam niet conform best practices. Meestal worden voor dit doel id’s gebruikt. Het kan daarom aangenomen worden dat deze problematische situatie in dit geval een non-issue is. Wel is het iets waar rekening mee gehouden moet worden, en het is deze waarschijnlijk op te lossen door data op te schonen, bijvoorbeeld met OpenRefine.

Het moet worden opgemerkt, dat er geen technische oplossing is gevonden voor wat betreft conflicterende data, de actualiteit van data en het verwijderen van niet-actuele data. Als tijdelijke workaround wordt de “data warehouse” aanpak aangeraden, waarbij bij elk conversie van data de desbetreffende triple store leeg gemaakt wordt, en op nieuw aangevuld. Deze oplossing wordt vaak toegepast bij ETL-processen en in batchmodus. Er zou geen storing moeten zijn op de bedrijfsprocessen als de aanpak in de nachturen op een server wordt uitgevoerd.

4.3.2 Deelconclusie

Op basis van de resultaten en de uitgevoerde analyse is het te concluderen dat het prototype, gerealiseerd op basis van de eerder opgestelde ontwerpen voldoet aan een grote deel van de gestelde eisen (zie Bijlage 16). De casus is succesvol nagemaakt en de resultaten bleken acceptabel te zijn. Het proces om tot dit resultaat te komen is uniform vanwege het gebruik van de templates. Het proces is geautomatiseerd op twee gebieden. Ten eerste is een transformatie van een bestand geautomatiseerd nadat er een template is aangemaakt; ten tweede is het mogelijk om transformatie van meerdere bestanden tegelijkertijd uit te voeren. De verwachting is dat deze wijze van werken efficiënter is qua tijd, maar ook qua foutgevoeligheid. De bewerkingen die nogal repetitief zijn kunnen vanaf nu geautomatiseerd worden, waardoor de kans op typefouten kleiner wordt.

Het is echter niet gelukt om een oplossing te bieden voor de kwesties: de actualiteit van data, de conflicterende data en het verwijderen van niet-actuele data. De hiervoor voorgestelde tijdelijke

oplossing is een datawarehouse approach, waarbij de oude data verwijderd wordt, voordat deze is aangevuld met nieuwe data.

4.4 Conclusie

"Hoe kan een framework voor data integratie binnen de context van het lectoraat DIO worden gerealiseerd op basis van semantische web technologieën?"

Uit het literatuuronderzoek is gebleken dat het semantische web en desbetreffende technologieën nog in een vroeg stadium van ontwikkeling zijn. Zo zijn er veel tools aanwezig om met het semantische web aan de slag te gaan; echter ontbreekt het aan tools om het semantische web te onderhouden. Langzaamaan worden er onderzoeken uitgevoerd op het gebied van data integratie van semantische data. Zo worden de potentiële issues in kaart gebracht, en worden er eerste pogingen gedaan om tools te ontwikkelen om deze op te lossen. Deze tools kunnen nu al grotendeels gebruikt worden, maar ze zijn nog niet helemaal verfijnd.

Het lectoraat DIO is bezig om semantische web technologieën toe te passen voor het doel van de valorisatie van data. Tijdens dit proces is het duidelijk geworden dat er een aantal issues zijn. Zo is het proces van data integratie gedecentraliseerd en verloopt het volgens niet-gestandaardiseerde wijze. Hierbij worden veel handmatige bewerkingen gedaan, wat veel manuren kost en foutgevoelig is. Daarom is het wenselijk om dit proces zo veel mogelijk te automatiseren. Ook is het wenselijk om dit proces simpel te maken, zodat deze door een niet-ICT'er uitgevoerd kan worden. Daarbij is het wel logisch dat zo'n proces eerst door een expert ontwikkeld wordt. De gebruikte datasets kunnen soms zeer groot worden, waarbij de grens op dit moment rondom 500MB ligt; het framework moet dat kunnen ondersteunen. Bovendien is de aangeleverde data vaak "vies" en niet-genormaliseerd, waardoor deze eerst opgeschoond moet worden voor het gebruik. Het uiteindelijk resultaat moet een semantische dataset zijn, dat bruikbaar is voor vervolganalyses.

Op basis van het theoretisch onderzoek en de opgestelde eisen, is er een ontwerp gemaakt. Dit ontwerp beschrijft het toepassen van OpenRefine en RML-Processor als een data integration framework. Op basis van dit ontwerp is er een prototype gebouwd. Dit prototype dient ter validatie van het gerealiseerde framework. Bovendien is het resulterend dataset gecontroleerd op correctheid. Uit deze tests bleek dat de conversie grotendeels op een correcte wijze verloopt. Echter zijn er nog een aantal aandachtpunten. Allereerst zijn de resulterende objecten van triples opgeslagen als Strings, al is de aangeleverde data van een ander type. Verder kan het framework niet goed omgaan met Nederlandse tekens; deze moeten in eerdere stappen omgezet worden naar gewone tekens. Desalniettemin kan dit framework toegepast worden om het proces van opschonen van data en dataconversie efficiënter te laten verlopen.

Kortom, uit het onderzoek blijkt dat een combinatie van OpenRefine en RML-Processor toegepast kan worden binnen lectoraat DIO, om het proces van vertalen van niet-semantische data naar semantische data, en het integreren hiervan in een triple store efficiënter te laten verlopen.

4.4.1 Vergelijking met ander onderzoek/theorie

RML-Processor – een onderdeel van data integration framework – is voorgesteld in een paper *"RML: A Generic Language for Integrated RDF Mapping of Heterogeneous Data"* (Dimou, et al., 2014) als oplossing van problemen voortkomend uit integratie van heterogene data. Deze paper was vooral gericht op de theoretische basis voor deze aanbeveling. Tijdens dit onderzoek is deze theorie ook praktisch toegepast in de ZeelandPas case. Conclusies uit dit onderzoek zijn vergelijkbaar, namelijk dat RML-Processor goed inzetbaar kan zijn bij data integratie. In de eerder genoemde paper wordt

een aantal mogelijke verbeteringen benoemd, dat aangevuld kan worden door verbeteringen zoals beschreven in dit rapport.

4.4.2 Suggesties voor vervolgonderzoek

Hoewel data integration framework succesvol is ingezet in de ZeelandPas case en waarschijnlijk ook in de bredere context ingezet kan worden, zijn er een aantal verbeterpunten dat door een vervolgonderzoek opgenomen zou kunnen worden.

Ten eerste is er een kwestie van data types. Zoals eerder in het rapport vermeld, werden integers omgezet naar Strings. Dat kan een negatieve invloed hebben op vervolganalyses. Zo wordt het moelijker om berekeningen uit te voeren op basis van deze geconverteerde gegevens. Het effect hiervan is dat een aritmetische bewerking ““1” + ”1”” als resultaat “11” oplevert (samenvoegen van strings in plaats van optelling van getallen). Dit probleem kan weliswaar genegeerd worden door middel van SPARQL-query, echter zou het wenselijker zijn als het niet nodig is.

Ten tweede is er een kwestie van het omgaan met al aanwezige data. Wanneer er voor de tweede keer een data set gegenereerd wordt, en toegevoegd wordt aan een al bestaande een, dan worden deze samengevoegd. Data worden niet gedupliceerd; echter is het mogelijk dat er niet-actuele data aanwezig kan zijn in de triple store. Dit probleem is op te lossen door alles te verwijderen, en vervolgens een nieuwe set in te laden. Deze aanpak is alleen niet efficiënt, vooral naarmate de hoeveelheid data zal stijgen. Een betere oplossing zou een complexe ontologie zijn met vastgestelde ranges en domeinen, waarin de niet actuele gegeven overgeschreven worden. Het toevoegen en analyseren van metadata zou een andere optie kunnen zijn. Voorlopig lijkt de “datawarehouse approach” voldoende, maar het is wel iets om rekening mee te houden.

Ten derde lijkt het verstandig om een overkoepelend GUI te bouwen, die de OpenRefine en RML-Processor aan elkaar zouden “lijmen”. In het huidige framework moeten de databestanden handmatig verplaatst worden. Het resultaat moet vervolgens ook handmatig ingeladen worden. Een GUI die deze onderdelen zou overkoepelen, zal een positief impact hebben op user experience. Verder maakt zo’n aanpak het mogelijk om extra elementen/modules toe te voegen onder de motorkap, zonder dat het complexer wordt voor de gebruiker. Bijvoorbeeld een extra module die andere databestanden kan converteren. De gebruiker zou dan niet hoeven na te denken over de tool die hij nodig heeft; de software kan dat zelf bepalen.

4.4.3 Tot besluit

De aanbeveling is om dit data integration framework als prototype te gebruiken. Uit het onderzoek blijkt dit framework een toegevoegde waarde te kunnen leveren aan de bedrijfsprocessen van het lectoraat DIO. Uiteraard moet er rekening gehouden worden met het feit dat de software niet compleet is, en dus voor uitbreiding vatbaar. De aanbeveling hierbij is om dit framework bij andere cases toe te passen, en waar nodig aanpassingen te doen aan de software. Verder is de aanbeveling om vervolgonderzoek uit te voeren, vooral voor wat betreft de kwestie van datatypes en het omgaan met al aanwezige data.

5 Literatuur

Allemang, D., & Hendler, J. (2011). *Semantic Web for the Working Ontologist.* Waltham: Morgan Kaufmann.

Arthur, L. (2013, Augustus 15). *What is Big Data?* Opgehaald van Forbes: http://www.forbes.com/sites/lisaarthur/2013/08/15/what-is-big-data/

Baker, M. (2009, Oktober 2). *What is a Software Framework? And why should you like 'em?* Opgehaald van Cimetrix: http://info.cimetrix.com/blog/bid/22339/What-is-a-Software-Framework-And-why-should-you-like-em

Dictionary.com. (2015). *semantic*. Opgehaald van Dictionary.com: http://dictionary.reference.com/browse/semantic

Diestel, R. (2010). *Graph Theory.* Heidelberg: Springer-Verlag.

Digital Enterprise Research Institute. (sd). *Refine*. Opgehaald van RDF Refine: http://refine.deri.ie/

Digital Enterprise Research Institute. (sd). *Refine*. Opgehaald van RDFExport: http://refine.deri.ie/rdfExport

Dimou, A., Sande, M. V., Colpaert, P., Verborgh, R., Mannens, E., & Walle, R. V. (2014). *RML: A generic Language for Integrated RDF mappings of heterogeneous data.* Ghent: Ghent University - iMinds - Multimedia Lab.

Embley, D. W., & Thalheim, B. (2011). *Handbook of Conceptual Modeling.* Berlin: Springer.

Frost, A. (2013). *Defining knowledge, information, data*. Opgehaald van KMT: http://www.knowledge-management-tools.net/knowledge-information-data.html

Gagnon, M. (2007). *Ontology-Based Integration of Data Sources.* Quebec: Defence R&D Canada.

Hendler, J. (2009). Web 3.0 Emerging. *Computer*, 111-113.

Hohpe, G., & Woolf, B. (2007). Enterprise Integration Patterns. Addison-Wesley.

Hohpe, G., & Woolf, B. (2007). Enterprise Integration Patterns. Addison-Wesley.

Hull, R. (1997). *Managing Semantic Heterogeneity in Databases.* New York: Bell Laboratories.

IBM. (2015). *What is data integration?* Opgehaald van IBM: http://www-01.ibm.com/software/data/integration/

Information Sciences Institute, University of Southern California. (2014). *Karma*. Opgehaald van Information Sciences Institute: http://www.isi.edu/integration/karma/

Lemire, D. (2012, September 4). *Publicly available large data sets for database research*. Opgehaald van Lemire.me: http://lemire.me/blog/archives/2012/03/27/publicly-available-large-data-sets-for-database-research/

Microsoft. (2013, Juli 12). *Description of the database normalization basics*. Opgehaald van Microsoft Support: http://support.microsoft.com/kb/283878?wa=wsignin1.0

OpenRefine. (2015, Februari 23). *Welcome*. Opgehaald van OpenRefine: http://openrefine.org/

Oracle. (2014). *Overview of Extraction, Transformation, and Loading*. Opgehaald van Oracle Help Center: http://docs.oracle.com/cd/B19306_01/server.102/b14223/ettover.htm

Price, R. (2004, Oktober 1). *What is an RDP Triple?* Opgehaald van RoberPrice: http://www.robertprice.co.uk/robblog/2004/10/what_is_an_rdf_triple_-shtml/

Semantic Web. (2010, Augustus 18). *History and Context*. Opgehaald van Semanticweb: http://www.semanticweb.rs/Article.aspx?iddoc=32&id=65&lang=2

Slaghuis, L. (2009, februari 18). *Semantic Web - Hoe werkt het nou echt?* Opgehaald van Frankwatching: http://www.frankwatching.com/archive/2009/02/18/semantic-web-hoe-werkt-het-nou-echt/

Technopedia. (2015). *Software Framework*. Opgehaald van Technopedia: http://www.techopedia.com/definition/14384/software-framework

University of Mannheim. (2014, Februari 13). *About*. Opgehaald van LDIF - Linked Data Integration Framework: http://ldif.wbsg.de/#about

W3C. (2009, Augustus 18). *SKOS Simple Knowledge Organization System*. Opgehaald van w3.org: http://www.w3.org/TR/skos-reference/

W3C. (2009). *W3C Semantic Web Frequently Asked Questions*. Opgehaald van W3C Semantic Web: http://www.w3.org/RDF/FAQ

W3C. (2012, September 27). *R2RML: RDB to RDF Mapping Language*. Opgehaald van W3.org: http://www.w3.org/TR/r2rml/#vocabulary

W3C. (2014, December 1). *ConverterToRdf*. Opgehaald van W3C: http://www.w3.org/wiki/ConverterToRdf

Worboys, M., & Duckham, M. (2004). *GIS: A Computing Perspective.* CRC Press.

Bijlage 1 Vragenlijst interview

Interview: Anton Bil

Technische vragen

- Hoe wordt data aangeleverd? (bestandtypen/services/etc.)
- Is er een identificatie systeem voor de leverancier van data?
    - Worden de bestanden met vaste namen opgeleverd (zo niet, kan dat gevraagd worden?)
        - Dat voor het doel van automatisch vaststellen van leverancier van data
- Moet er rekening gehouden worden met verschillende datum formaten (DMJ, MDJ enz.)?
- Is er een harde eis en/of voorkeur naar een bepaald programmeertaal?
- Wat moet het resultaat zijn van het framework?
    - Moet het een geconverteerd bestand opleveren of moet het data direct in een database ingevoerd worden?
- Wat is er al beschikbaar qua infrastructuur? (RDF store etc.)

Functionele vragen

- Is batch processing van databronnen wenselijk?
- Ligt de voorkeur aan van te voren definiëren van links tussen templates en databronnen, handmatig selecteren of combinatie van beiden?
- In hoeverre moet het proces geautomatiseerd worden?
- Moet het mogelijk zijn om het resultaat van integratie binnen de applicatie zichtbaar te maken of worden hiervoor andere tools voor gebruikt?
    - Bijvoorbeeld ter controle van data kwaliteit

Interview: Hans de Bruin

- Hoe wordt data aangeleverd? (bestandtypen/services/etc.)
- Is er een harde eis en/of voorkeur naar een bepaald programmeertaal?
- Hoe wordt het framework in volgende projecten toegepast?
- Door wie wordt het framework in toekomst gebruikt
- Wat is er al beschikbaar qua infrastructuur en hoe zal dat in de toekomst eruit zien? (RDF store etc.)

Interview: Anton Bil Resultaten

Technische vragen

Hoe wordt data aangeleverd? (bestandtypen/services/etc.)

Op dit moment wordt data opgeleverd als CSV en Excel bestanden. Maar in de toekomst kunnen ook andere formaten gebruikt worden, daarom is het handig als het framework deze ondersteunt. RDF kunnen verder ook gebruikt worden, gezien dat het uiteindelijk in een triplestore wordt ingeladen. **(REQ_001 en REQ_015)**

Is er een identificatie systeem voor de leverancier van data? Worden de bestanden met vaste namen opgeleverd?

Over het algemeen is het aan de naam van bestand te zien en zijn er afspraken gedaan om het zo te houden. Wel moet er rekening gehouden worden met de mogelijkheid dat de namen en datastructuur kunnen wijzigen, en moet zo'n bewerking binnen het framework mogelijk zijn. **(REQ_006)**

Moet er rekening gehouden worden met verschillende datum formaten (DMJ, MDJ enz.)?

Hier moet zeker rekening mee gehouden worden (en ook andere data typen) en moeten deze geconverteerd worden naar standard notatie dat in SPARQL gebruikt wordt. Wel is er over het algemeen duidelijk welk formaat er gebruikt wordt, dus is het handmatige identificatie simpel. **(REQ_009)**

Is er een harde eis en/of voorkeur naar een bepaald programmeertaal?

Op het gebied van programmeertalen is er geen harde eis. Wel wordt Java aangeraden i.v.m. integratie met vaak gebruikte software (zoals Google Refine en Fuseki)

Wat moet het resultaat zijn van het framework? Moet het een geconverteerd bestand opleveren of moet de data direct in een database ingevoerd worden?

Het moet mogelijk zijn om de data naar RDF store te sturen en inladen. Wel graag RDF bestanden als back-up genereren. **(REQ_003, REQ_004 en REQ_005)**

Wat is er al beschikbaar qua infrastructuur? (RDF store etc.)

Op dit moment wordt Fuseki gebruikt (deze draait op een server, maar er is ook een VM beschikbaar als testomgeving), soms wordt ook Virtuoso, als RDF store gebruikt. Ook is er een semantische media wiki aanwezig, dat gelinkt is met de triple store.

Functionele vragen

Is batch processing van databronnen wenselijk?

Dat is zeer wenselijk. Zo worden er diverse bestanden geleverd, waarop in een aantal stappen bewerkingen (aanvullen, verrijken, converteren) uitgevoerd moeten worden. Dat moet zo automatisch mogelijk verlopen, om menselijke fouten te voorkomen. Uiteindelijk moet het data op een server komen. **(REQ_002, REQ_003 en REQ_009)**

Ligt de voorkeur aan van te voren definiëren van links tussen templates en databronnen, handmatig selecteren of combinatie van beiden?

Van te voren definiëren, hoeft geen GUI te zijn (gebeurt op dit moment ook zonder), hoeft niet met anderen te delen zijn (dat valt buiten de scope en zijn hier andere software oplossingen voor). Wel is het belangrijk dat menselijke handelingen zoveel mogelijk worden geautomatiseerd om fouten te voorkomen. Op dit moment is het bijvoorbeeld nodig om templates die bij google Refine gebruikt worden handmatig te kopiëren. **(REQ_006)**

In hoeverre moet het proces geautomatiseerd worden?

Zelf op knop drukken, het is niet nodig om deze vanzelf te laten starten. De stappen die vervolgens uitgevoerd moeten worden (kopiëren van bestanden, aanroepen van conversie enz.) moeten wel zoveel mogelijk automatisch. **(REQ_010)**

Moet het mogelijk zijn om het resultaat van integratie binnen de applicatie zichtbaar te maken of worden hiervoor andere tools voor gebruikt?

Graag log bijhouden, zodat het duidelijk is wat er precies gedaan werd. Graag meerdere niveaus van diepte (bijvoorbeeld algemeen, warnings, errors) dat van te voren ingesteld kunnen worden. **(REQ_011, REQ_013)**

Aanvullende notities

- In de werkelijke situatie wordt er gewerkt met grote bestanden. Beschikbare tools laden het geheel bestand in voordat deze verwerkt worden. Dat is onwenselijk, omdat het geheugen van de computers overbelast raakt.**(REQ_007 en REQ_008)**
- Deel van data wordt ongestructureerd geleverd (tekst velden). Daarin is woord volgorde etc. niet altijd hetzelfde. **(REQ_013)**
- **Volume**, **variety**, velocity, veracity (4V's) is een belangrijk thema bij Big Data (en dus ook de opdracht). Zoals het nu is, speelt vooral volume en variety een grote rol. **(REQ_007 en REQ_009)**
- Op dit moment worden de werkzaamheden omtrent data integration door 1 persoon uitgevoerd, maar dat gaat in de toekomst anders worden. Zo wordt er een content-manager aangewezen die het procedure moet kunnen uitvoeren. Je moet je dus voorstellen dat de procedure in eerste instantie door een expert uitgetest wordt maar dat de content manager op den langen duur het onderhoud overneemt. **(REQ_009 en REQ_014)**
- Op dit moment wordt OpenRefine gebruikt om data te verrijken en op te schonen. Hierin zijn wel aantal tijdrovende stappen te benoemen. Zo moet elk bestand afzonderlijk voorzien worden van een template. Een template in dit geval bestaat uit: prefix, row template, row separator en een suffix. Deze zijn in een media wiki opgeslagen als tekst, dat stuk voor stuk gekopieerd naar het template moet worden. Dat is niet wenselijk, omdat er elk keer per bestand 4 kopieer acties uitgevoerd moeten worden. **(REQ_016)**
- Een job wordt eerst voorbereid en getest door een expert. Deze stelt vervolgens een handleiding voor, zodat het onderhoud door een content manager uitgevoerd kan worden. Van een content manager kan je verwachten dat deze verstand heeft van inhoud, bestanden kan verplaatsen en vooral met GUI werkt, maar voor de rest weinig verstand van ICT **(REQ_014)**
- Elk keer dat er een job uitgevoerd wordt, wordt er een nieuwe triplestore aangemaakt, en de oude wordt verwijderd

Interview: Hans de Bruin Resultaten

Hoe wordt data aangeleverd? (bestandtypen/services/etc.)

Het kan van alles zijn. Het kan een database zijn, het kan een spreadsheet zijn, web zelf, internet site, web-services, maakt niet uit, alle data soorten kunnen er aanwezig zijn. Ook nog vaak is deze data incompleet. In geval van ZeelandPas proberen we data te verrijken met weerinformatie, maar het kan zo zijn dat er op sommige gebieden geen weerinformatie beschikbaar is. Ook moet er rekening gehouden worden met conflicterende data. Zo kan er in een bron staat dat er zon scheen en in een ander dat er geregend heeft.

Is er een harde eis en/of voorkeur naar een bepaald programmeertaal?

Het moet in ieder geval in semantische web kunnen gebeuren en dat betekent dus dat je met SPARQL aan de slag moet, en dat kan je dan in een ander programmeertaal embedden, zoals PHP of Java. Andere mensen moeten er verder ook mee aan de slag kunnen, dus liever geen exotische programmeertalen.

Hoe wordt het framework in volgende projecten toegepast?

Waar we mee te maken hebben is, met een issue dat we kwantitatieve gegevens hebben via sensoren of wat dan ook, waar we de gegevens uit krijgen. Die gegevens zouden we in relatie met

elkaar gaan brengen en daar weer zinvolle dingen over zeggen, dus we willen de verbanden uit gaan trekken. En wat we met framework willen bereiken, is dat we dat op een eenvoudige manier kunnen doen. Dus dat we een aantal standaard aanpakken hebben, dat het niet uitmaakt of het een spreadsheet of sensor data is. Essentie zit niet in het soort formaten, er zijn al aantal tools die het voor je doen. Waar het om gaat is, hoe kan ik er voor zorgen dat ik twee databronnen heb, hoe zorgt ik ervoor dat ze aan elkaar verknoopt worden. Dat zal niet altijd 1 op 1 gaan. Hier gaat het om, het merging van data. En bij een update, als er een nieuwe versie bij komt? Wat gooi ik dan weg en wat moet er weer bij komen. Want het kan niet zo zijn, dat we steeds data op data stappelen.

Ook is de uitgangspunt dat we gebruik gaan maken van EMont en SKOS gaan we ook vanuit.

Door wie wordt het framework in toekomst gebruikt

Door het raamwerk zelf, het liefst willen we dat het automatisch gebeurt. Als het blijkt dat het dusdanig complex is, dat het niet geautomatiseerd kan, dat kan ook en zal het op ander manier kunnen.

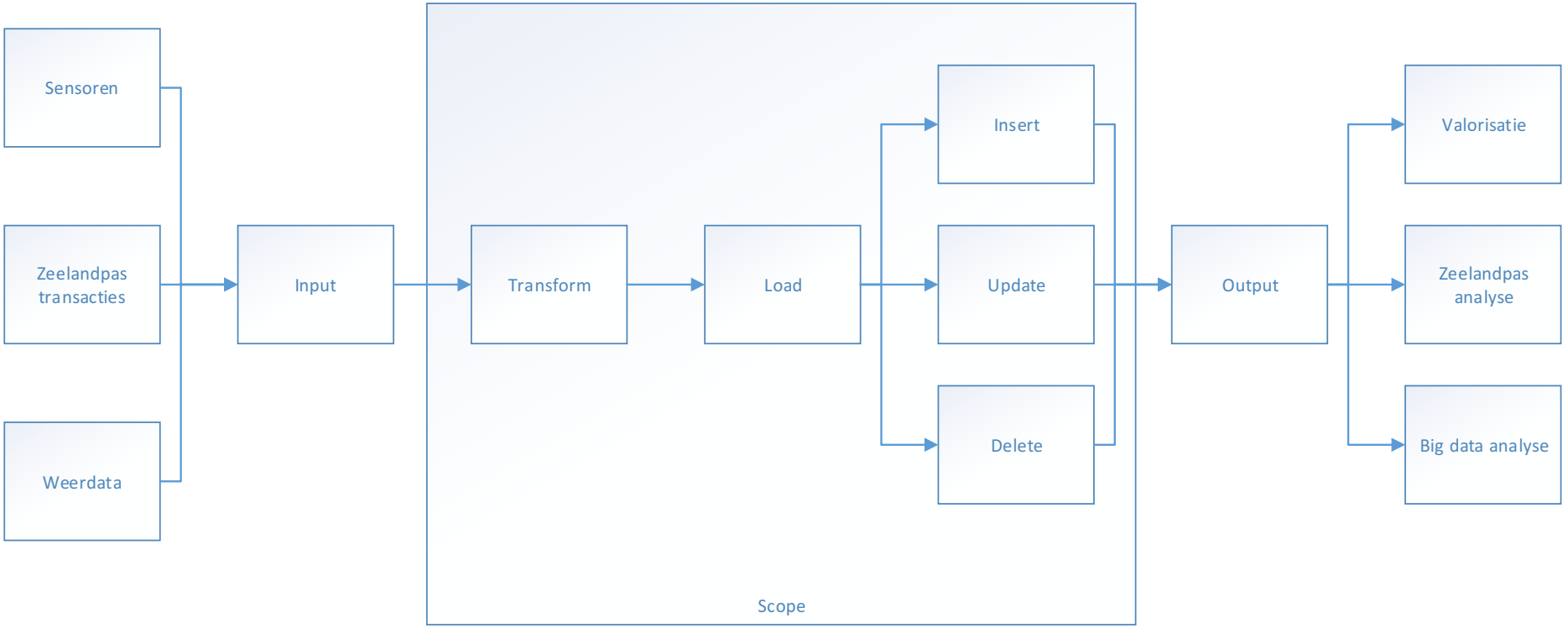
Wat is er al beschikbaar qua infrastructuur en hoe zal dat in de toekomst eruit zien? (RDF store etc.)

Het uitgangspunt is altijd een triple store en semantic wiki. Triple store wordt op locatie gehouden. SMW, waar we nu mee bezig zijn werkt ook op een triple store.

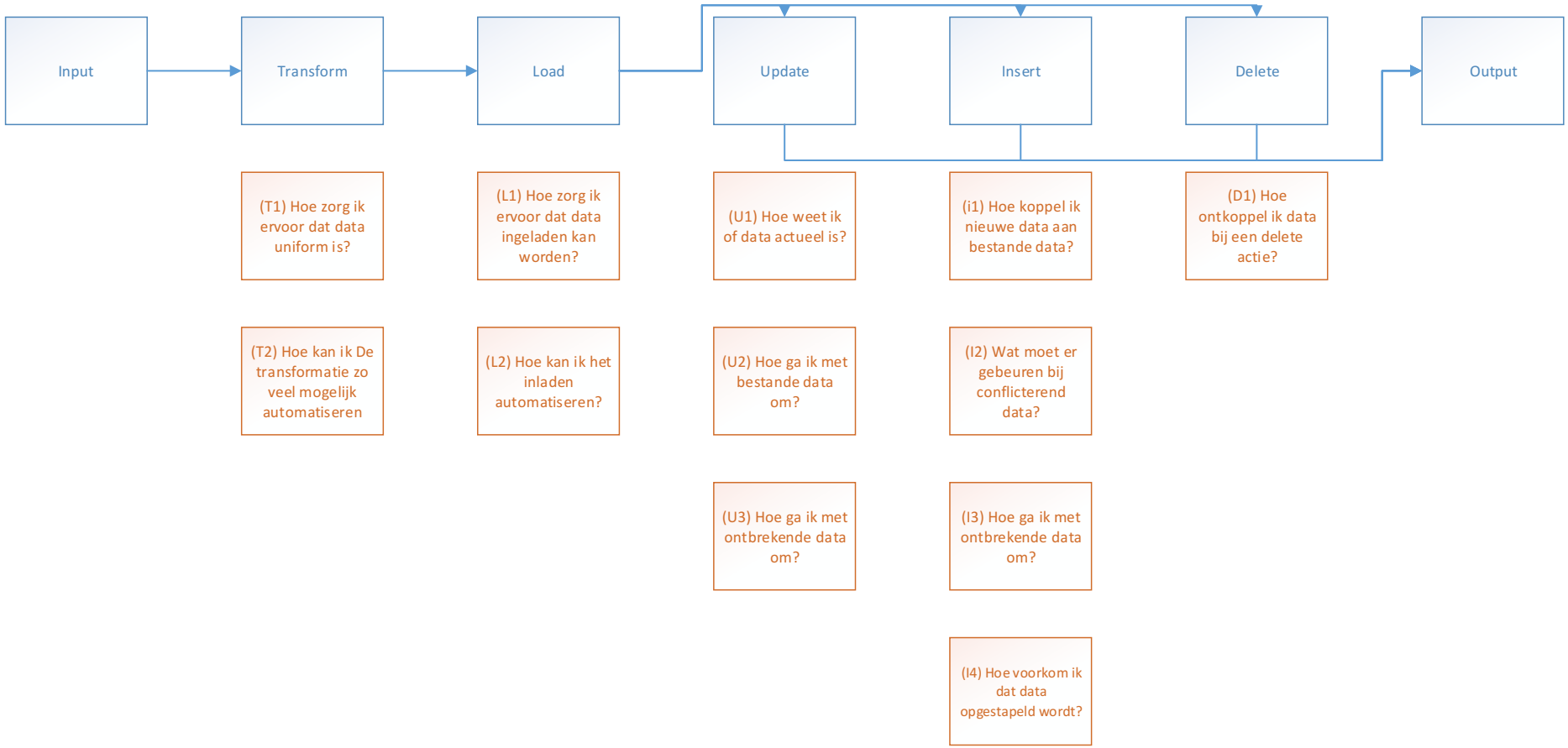
Bijlage 2 Requirements – MoSCoW

ID	Prioriteit	Eis	Status
REQ_001	Must	Framework moet data bestanden kunnen converteren naar RDF	To do
REQ_002	Should	Framework zou een optie moeten bieden om meerdrere bestanden achter elkaar unattended te converteren	To do
REQ_003	Should	Framework zou moeten geconverteerde bestanden naar een server sturen	To do
REQ_004	Should	Framework zou moeten geconverteerde bestanden in een RDF database laden	To do
REQ_005	Should	Framework zou moeten geconverteerde bestanden als backup bijhouden	To do
REQ_006	Must	Framework moet een optie bieden om templates te koppelen met bestanden	To do
REQ_007	Should	Framework zou moeten grote bestanden (tot 500 MB) moeten ondersteunen	To do
REQ_008	Could	Framework zou kunnen de conversie per row kunnen uitvoeren, om geheugen te besparen	To do
REQ_009	Must	Templates moeten ervoor zorgen dat de data op een uniforme wijze geformateerd wordt	To do
REQ_010	Won't	Framework zal niet in de achtergrond werken en zonder menselijke input starten/converteren	To do
REQ_011	Should	Framework zou moeten een log-bestand genereren	To do
REQ_012	Should	Framework zou het volgende logging niveaus moeten kennen: error, warning, info	To do
REQ_013	Must	Framework moet een optie bieden om text velden te splitsen in afzonderlijke velden	To do
REQ_014	Should	Framework zou moeten door een persoon met minimale ICT ervaring onderhoudbaar moeten zijn	To do
REQ_015	Should	Framework zou flexibel genoeg moeten zijn om in de toekomst ook andere data bestanden te ondersteunen	To do
REQ_016	Should	Het zou mogelijk moeten zijn om template op te slaan om deze te hergebruiken	To do

Bijlage 3 Scope



Bijlage 4 Problem mapping I



Bijlage 5 Problem mapping II

Ter verduidelijking van in de Problem Mapping benoemd potentiële problemen, zijn deze hieronder nader toegelicht. Elk potentiële probleem is voorzien van code, deze bestaat uit domain van het probleem (T – Transform, L – Load, U – Update, I – Insert, D – Delete), en het volgorde nummer.

Transform problemen

Met transform problemen worden er alle issues bedoeld, dat zich kunnen voordoen tijdens het transformeren van data, ten verbetering van datakwaliteit.

T1 – Hoe zorg ik ervoor dat data uniform is?

De aangeleverde data zal op verschillende formaten aangeleverd worden. Een goed voorbeeld hiervan is “datum” dat meerdere notaties kent. Om het analyseren van data mogelijk te maken, moet deze data uniform zijn. Deze handelingen gebeuren nu handmatig en zijn niet gestructureerd. Het is wenselijk dat hier een gestandaardiseerde oplossing voor komt.

T2 – Hoe kan ik de transformatie zo veel mogelijk automatiseren?

De aangeleverde data wordt vaak verspreid aangeleverd. Dat wil zeggen dat data in meerdere bestanden verdeeld worden. Dat betekent dat transformatie op elk van deze bestanden moet plaatsvinden. Een handmatige bewerking, al is deze gestructureerd, is monotoon voor de gebruiker, wat fouten kan veroorzaken. Het is wenselijk, om het proces van transformatie zo veel mogelijk te automatiseren.

Load Problemen

Met load problemen worden er alle issues bedoeld, dat zich kunnen voordoen tijdens het (voorbereiden op) inladen van data in de database.

L1 – Hoe zorg ik ervoor dat data ingeladen kan worden?

De aangeleverde data wordt op verschillende wijzen aangeleverd. Denk hierbij aan verschillende soorten bestanden, databases of services. Een triple store kan van zichzelf een XML, TTL of JSON inladen, maar de aangeleverde data is meestal van een ander formaat. Om het inladen mogelijk te maken, moet de data geconverteerd worden naar een ondersteund formaat.

L2 – Hoe kan ik het inladen automatiseren?

De aangeleverde data wordt vaak verspreid aangeleverd. Dat wil zeggen dat data in meerdere bestanden verdeeld worden. Deze handelingen worden op dit moment noodgedwongen achter elkaar handmatig uitgevoerd. Het is wenselijk, om het proces van inladen zo veel mogelijk te automatiseren.

Update Problemen

Met update problemen worden er alle issues bedoeld, dat zich kunnen voordoen tijdens het updaten of overschrijven van bestaande data.

U1 – Hoe weet ik of data actueel is?

Op dit moment is er geen manier om erachter te halen of al ingeladen data actueel is. Het is daardoor onmogelijk om te weten of er een update plaats moet vinden of niet. Als effect, wordt er op ongestructureerde wijze bepaald of data geüpdatet moet worden. De vraag is of dat wenselijk is en of er betere alternatieven beschikbaar zijn.

U2 – Hoe ga ik met bestande data om?

Een belangrijke vraag dat gesteld moet worden bij een update actie is, hoe wordt er met bestande data omgegaan? Data dat niet vaak veranderd - zoals landen of gemeenten - hoeven vaak niet aangepast worden. Echter wanneer deze als data aangeleverd zijn (via een bestand of service), worden de gegevens in een triple store verwijderd, en op nieuw ingeladen. Ook moet er rekening gehouden worden met gegevens, waar maar een bepaald aantal instanties van mogelijk zijn.

U3 – Hoe ga ik met ontbrekende data om?

De aangeleverd data kan soms ontbrekende gegevens bevatten. Dat kan verschillende oorzaken hebben, zoals menselijk of technisch fout. Het voorkomen hiervan, valt buiten de scope van het project, het framework moet echter hier rekening mee houden. Wat moet er gebeuren in een situatie, waarbij een dataset dat voor update gebruikt wordt, ontbrekende data bevat? Deze vraag is vooral interessant in geval dat oude data deze gegevens wel bevat. Hier moet de data integration framework rekening mee houden.

Insert Problemen

Met insert problemen worden er alle issues bedoeld, dat zich kunnen voordoen tijdens het invoegen van nog niet in de database beschikbare data.

I1 – Hoe koppel ik nieuwe data aan bestande data?

De kracht van de data opslaan op een semantische wijze, is het zoeken naar verbanden tussen beschikbare data. Dat is alleen mogelijk, wanneer deze verbanden aanwezig zijn. In geval dat data ingeladen wordt, door een aantal bestanden achterelkaar in te voegen (per source), ontstaan er “data eilanden” tussen welk geen relaties aanwezig zijn. Hiermee wordt dus een grote deel van potentieel informatie niet benut. Het is natuurlijk mogelijk, om deze verbanden handmatig aan te maken, maar de vraag is of dat wenselijk is – het zou immers voor invoer fouten kunnen zorgen. Bovendien zou zo’n bewerking zeer ongestructureerd van aard zijn, wat in tegenstrijd is met het nut van framework. Daarom is het van groot belang om te onderzoeken, of hier een alternatief voor is.

I2 – Wat moet er gebeuren bij conflicterend data?

In geval dat er data toegevoegd wordt, moet er rekening gehouden worden met al aanwezige data. Wat moet er bijvoorbeeld gebeuren wanneer er aan een persoon tweede “huidige woonadres” toegekend krijgt? De te realiseren data integration framework moet hier rekening mee houden.

I3 – Hoe ga ik met ontbrekende data om?

Het kan gebeuren, dat er door menselijk of technisch fout ontbrekende data aanwezig is. Wat moet er in zo’n situatie gebeuren? Moet de framework hier een melding van maken, moet zo’n invoeging mogelijk zijn en zo ja – in wat voor vorm?

I4 – Hoe voorkom ik dat data opgestapeld wordt?

Het toevoegen van data betekent dat er een nieuwe instantie van data aangemaakt wordt in een triple store. Dat zorgt voor dubbele data, wat foutieve resultaten zou kunnen opleveren bij data analyse. Er moet onderzocht worden hoe het framework hiermee rekening moet houden.

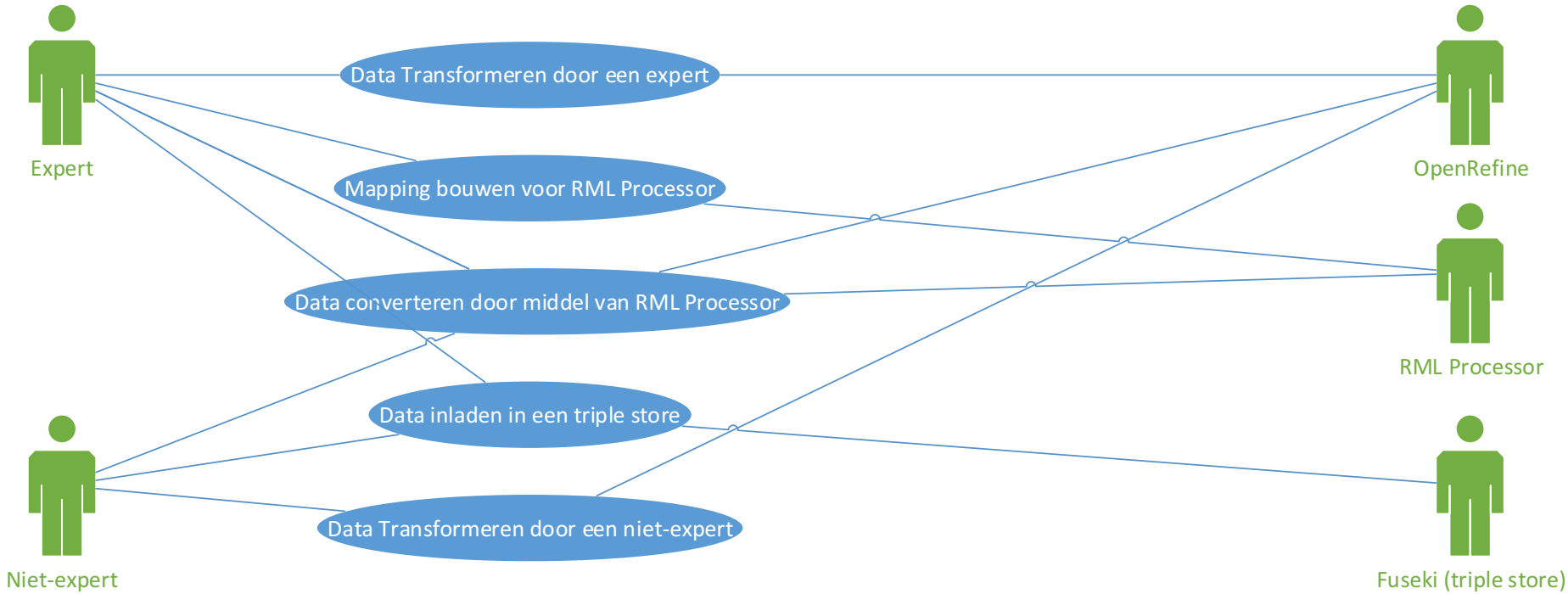
Delete Problemen

Met delete problemen worden er alle issues bedoeld, dat zich kunnen voordoen tijdens het verwijderen van in de database beschikbare data.

D1 – Hoe ontkoppel ik data bij een delete actie?

De verwachting is, dat de triple store een samenvoeging van verschillende data sets zal worden. Op deze manier is het mogelijk, om via data analyse nieuwe verbanden te ontdekken. Dat betekent wel dat data aan elkaar gekoppeld wordt. Maar dat kan problemen opleveren in geval dat deel van de data verwijderd moet worden. Bijvoorbeeld, het is goed mogelijk dat weerdata uit aantal verschillende bronnen gehaald kan worden (misschien is er geen bron dat weergegevens bevat voor alle nodige locaties, waardoor er een aggregatie van verschillende bronnen gebruikt wordt). Daardoor is het mogelijk, dat twee (of meer) bronnen iets zeggen over weer op een bepaald plek. Het verwijderen van data uit de eerste bron (bijvoorbeeld, omdat deze niet meer vertrouwd wordt), zou ook in het voorbeeld benoemd data verwijderen. Dat is niet wenselijk en het framework moet hier een oplossing voor bieden.

Bijlage 6 Use cases



Use case 1 – Data Transformeren door een expert

Description

Deze use case beschrijft het proces van transformeren van data door een expert.

Primary Actor

Expert

Precondition

- Actor heeft toegang tot data, dat is aangeleverd in een van de volgende formaten: TSV, CSV, XLS, XLSX, XML, RDF/XML, JSON, Google Spreadsheets, RDF N3-triples
- Actor is in bezit van OpenRefine, of kan een toegang krijgen via inter/intranet

Postcondition

Afhankelijk van keus van de actor, is er een RDF/XML, TTL of CSV gegenereerd.

Main Success Scenario

1. Actor ontvangt data
2. Actor start OpenRefine, of maakt gebruik van al opgestarte server
3. Actor maakt een project aan
4. Actor controleert data import op correctheid
5. Actor bevestigt project
6. Actor voert benodigde transformaties uit
7. Actor slaat het template op voor later gebruik
8. Actor exporteert CSV bestand

Extensions

Wanneer het wenselijk is om het bestand te exporteren in RDF formaat

1. Actor voert acties 1 t/m 6 uit
2. Actor past het RDF Skeleton aan
3. Actor slaat het template op voor later gebruik
4. Actor exporteert een RDF/XML of TTL bestand

Use case 2 – Data Transformeren door een niet-expert

Description

Deze use case beschrijft het proces van transformeren van data door een niet-expert

Primary Actor

Niet-expert

Precondition

- Actor heeft toegang tot data, dat is aangeleverd in een van de volgende formaten: TSV, CSV, XLS, XLSX, XML, RDF/XML, JSON, Google Spreadsheets, RDF N3-triples
- Actor is in bezit van OpenRefine, of kan een toegang krijgen via inter/intranet
- Actor is in bezit van een template

Postcondition

Afhankelijk van keus van de actor, wordt er een RDF/XML, TTL of CSV gegenereerd.

Main Success Scenario

1. Actor ontvangt data en een bijbehorend template
2. Actor start OpenRefine, of maakt gebruik van al opgestarte server
3. Actor maakt een project aan
4. Actor controleert data import op correctheid
5. Actor bevestigd project
6. Actor past template toe
7. Actor exporteert CSV, RDF/XML of TTL

Use case 3 – Mapping bouwen voor RML processor

Description

Deze use case beschrijft het proces van opbouwen van een RML mapping

Primary Actor

Expert

Precondition

- Actor heeft toegang tot data, dat is aangeleverd in een van de volgende formaten: CSV, TSV, JSON, XML, relationele database
- Er is consensus bereikt voor wat betreft te gebruiken datamodel
- Actor is in bezit van RML Processor

Postcondition

Een mapping dat gebruikt kan worden door een RML processor.

Main Success Scenario

1. Actor analyseert het datastructuur van aangeleverd bron
2. Actor analyseert het te gebruiken datamodel
3. Actor maakt zich bekend met RML
4. Actor maakt een RML mapping
5. Actor test het mapping door deze uit te voeren
6. Actor slaat het RML mapping op voor later gebruik

Use case 4 – Data converteren door middel van RML processor

Description

Deze use case beschrijft het proces van data conversie, gebruikmakend van RML processor

Primary Actor

Expert of niet-expert

Precondition

- Databestanden, mapping en RML processor bevinden zich op het juiste locatie
- Mapping is eerder getest en goedgekeurd

Postcondition

Een .ttl bestand dat ingeladen kan worden in een triplestore.

Main Success Scenario

1. Actor controleert of databestanden, mapping en RML processor zich op de juiste locatie bevinden
2. Actor zoekt het bijbehorende batchbestand op
3. Actor voert het bijbehorende batchbestand uit
4. Actor controleert het gegenereerd bestand

Extensions

Wanneer er geen batch bestand beschikbaar is

1. Actor controleert of databestanden, mapping en RML processor zich op de juiste locatie bevinden
2. Actor opent command prompt of vergelijkbaar commandline applicatie
3. Actor navigeert naar de locatie van RML processor
4. Actor wijst naar de locatie van RML processor, het mapping en output locatie
5. Actor controleert het weergegeven output log
6. Actor controleert het gegenereerd bestand
7. Zo nodig, slaat de actor het uitgevoerde commando als batch bestand

Use case 5 – Data inladen in een triple store (Fuseki)

Description

Deze use case beschrijft het proces van het inladen van ttl bestand in een triple store (Fuseki)

Primary Actor

Expert of niet-expert

Precondition

- Er is een ttl bestand gegenereerd
- Triple store is opgestart
- Actor heeft toegang tot de bestand en de triple store

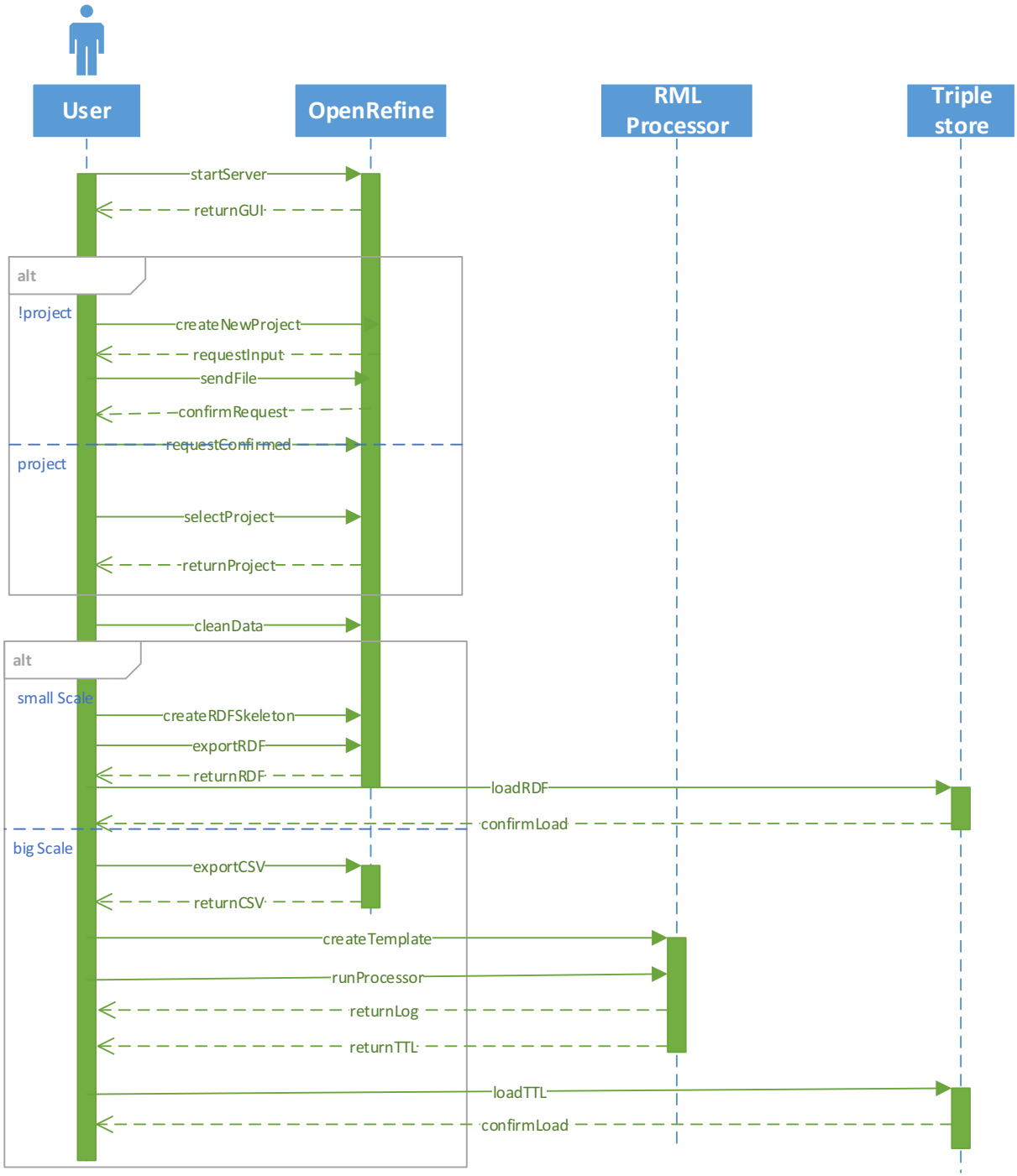
Postcondition

Data uit de ttl bestand is ingeladen in een triple store

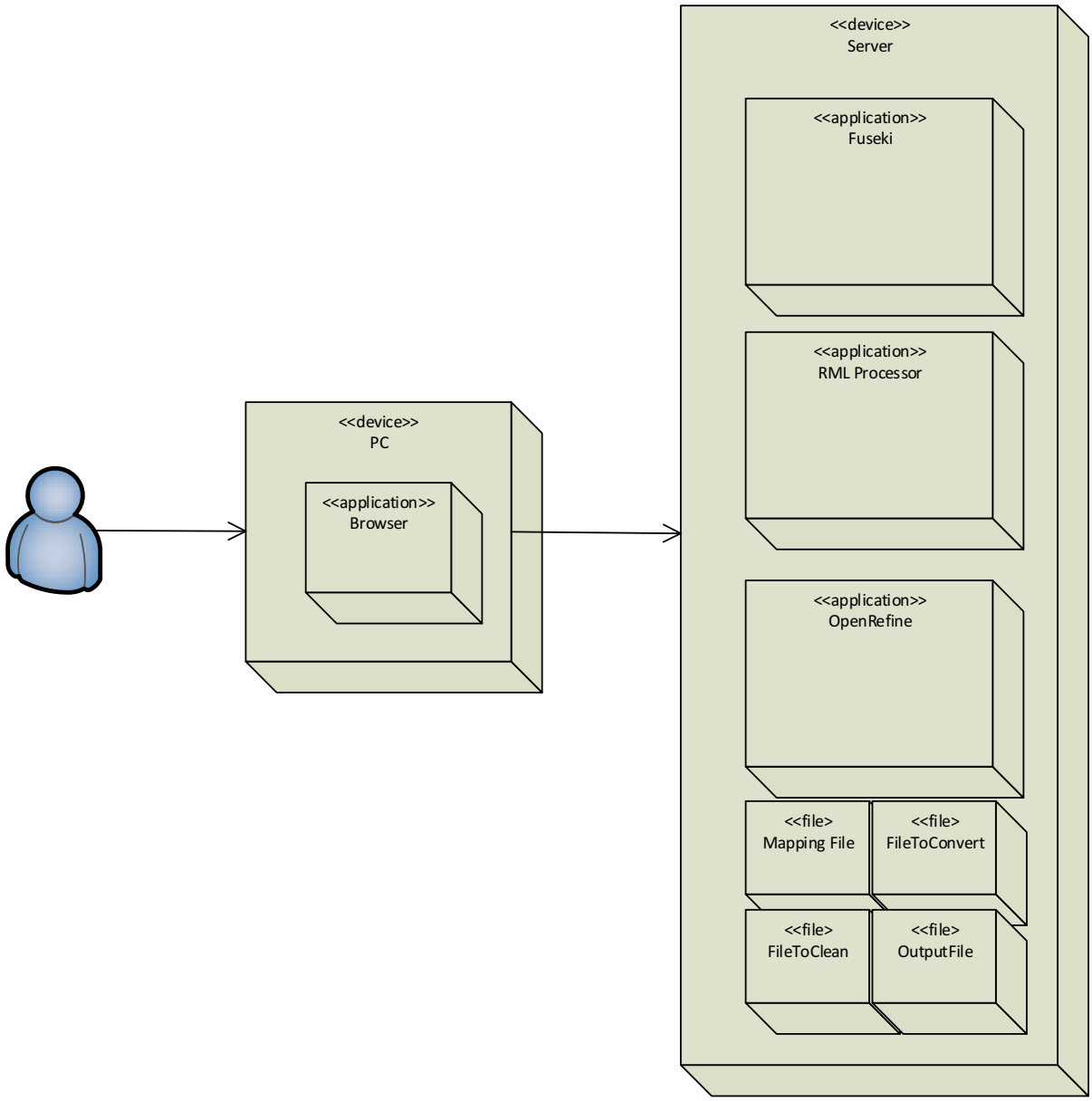
Main Success Scenario

1. Actor controleert of de triple store is opgestart
2. Actor controleert het ttl bestand
3. Actor navigeert naar de control panel van Fuseki
4. Actor selecteert de dataset
5. Actor selecteert het bestand om in te laden
6. Actor selecteert het graph
7. Actor drukt op knop “Upload”
8. Actor controleert het resultaat

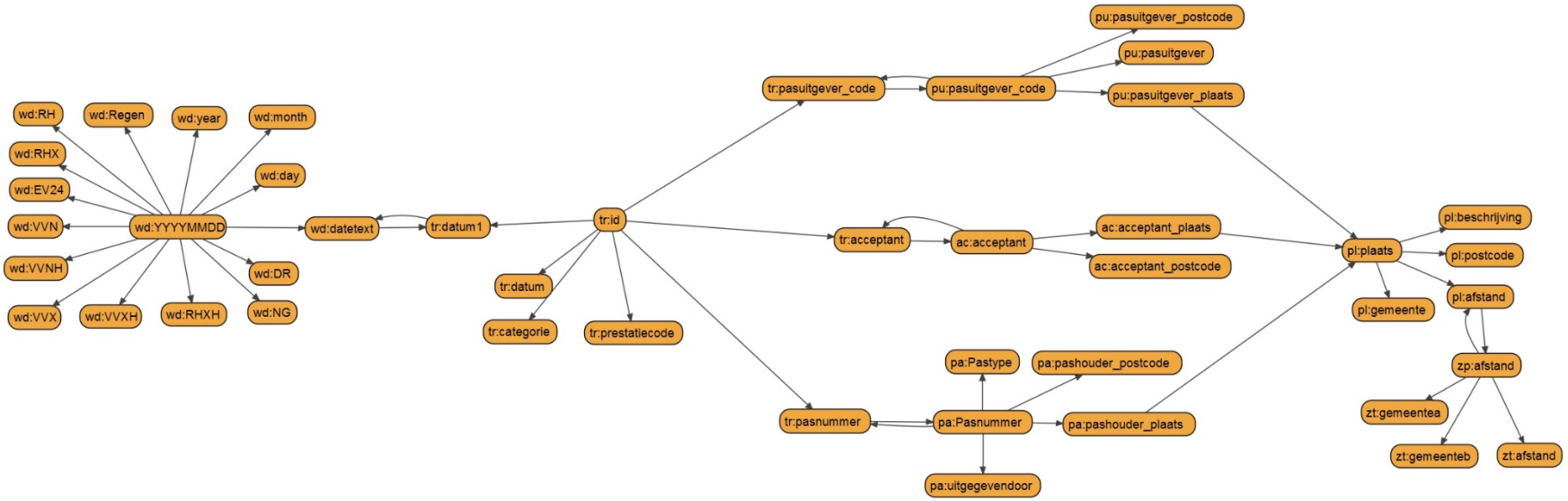
Bijlage 7 High-level sequence diagram



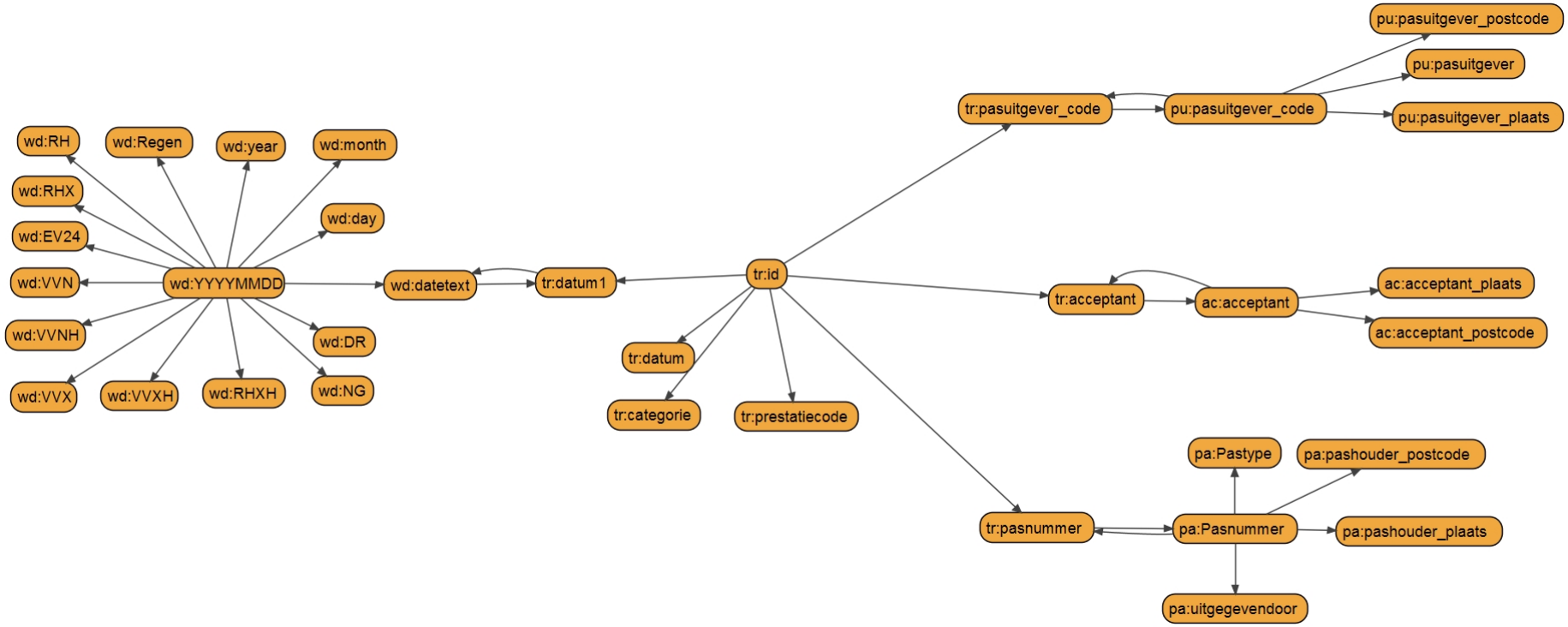
Bijlage 8 Deployment Diagram



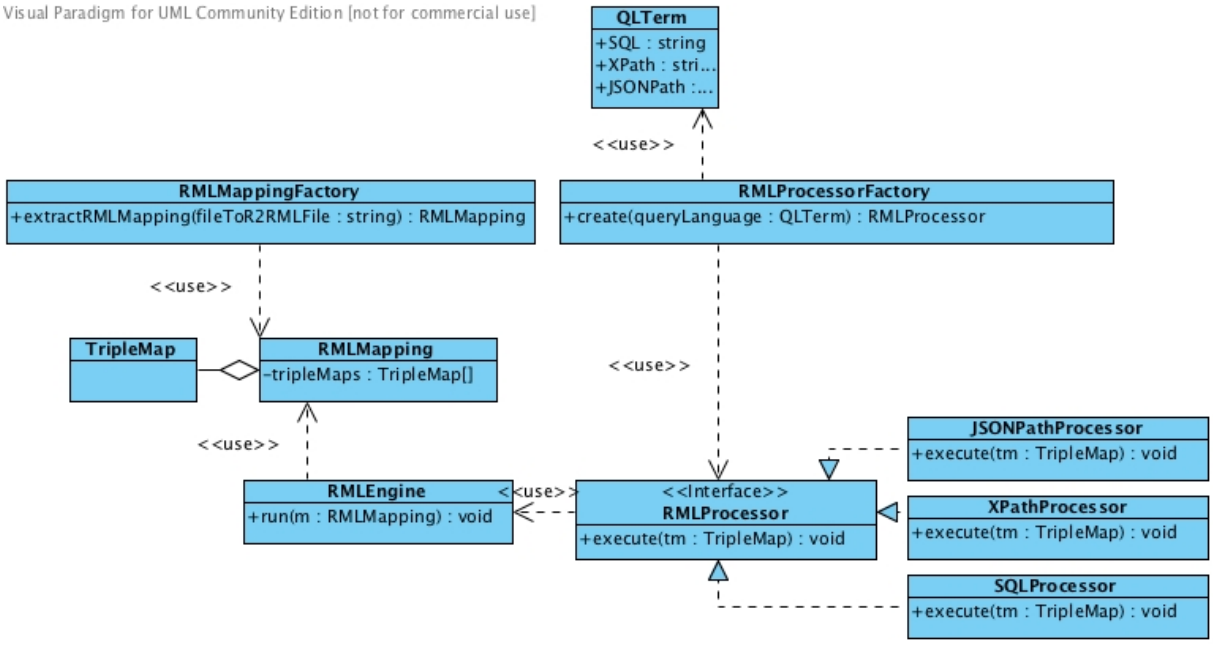
Bijlage 9 ZeelandPas Case Diagram Final



Bijlage 10 ZeelandPas Case Diagram 0.1



Bijlage 11 RML Processor Class Diagram



Bijlage 12 Proof of Concept

PoC – Kleinschalige mapping met 1 bron

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix rml: <http://semweb.mmlab.be/ns/rml#> .
@prefix ql: <http://semweb.mmlab.be/ns/ql#> .
@prefix tr: <http://example.org/dc/elements/1.1/transactie#> .
@prefix pa: <http://example.org/dc/elements/1.1/pas#>.
@prefix ac: <http://example.org/dc/elements/1.1/acceptant#> .
@prefix pu: <http://example.org/dc/elements/1.1/pasuitgever#> .

<#PasMapping>
  rml:logicalSource [
                rml:source "/afstuderen/experiment/input/transacties.csv";
    rml:referenceFormulation ql:CSV
  ];
  rr:subjectMap [
                rr:template "http://transactie.example.com/{Pasnummer}";
                rr:class pa:Pasnummer
  ];
  rr:predicateObjectMap [
                rr:predicate pa:Pastype;
                rr:objectMap [
                        rml:reference "Pastype"
                ]
        ];
        rr:predicateObjectMap [
                rr:predicate pa:uitgegevendoor;
                rr:objectMap [
                        rml:reference "Pasuitgever"
                ]
        ];
        rr:predicateObjectMap [
                rr:predicate pa:pashouder_postcode;
                rr:objectMap [
                        rml:reference "Pashouder postcode"
                ]
        ];
        rr:predicateObjectMap [
                rr:predicate pa:pashouder_plaats;
                rr:objectMap [
```

```
                        rml:reference "Pashouder plaats"
                ]
        ];
        rr:predicateObjectMap [
                rr:predicate pa:pashouder_land;
                rr:objectMap [
                        rml:reference "Pashouder land"
                ]
        ];
.
```

PoC – Kleinschalige mapping met een join binnen 1 bron

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix rml: <http://semweb.mmlab.be/ns/rml#> .
@prefix ql: <http://semweb.mmlab.be/ns/ql#> .
@prefix tr: <http://example.org/dc/elements/1.1/transactie#> .
@prefix pa: <http://example.org/dc/elements/1.1/pas#>.
@prefix ac: <http://example.org/dc/elements/1.1/acceptant#> .
@prefix pu: <http://example.org/dc/elements/1.1/pasuitgever#> .
@prefix zt:     <http://example.org/dc/elements/1.1/> .

<#PasMapping>
  rml:logicalSource [
                rml:source "/afstuderen/experimentParentSameCSV/input/transacties_modified.csv";
    rml:referenceFormulation ql:CSV
  ];
  rr:subjectMap [
                rr:template "http://transactie.example.com/{Pasnummer}";
                rr:class pa:Pasnummer
  ];
  rr:predicateObjectMap [
                rr:predicate pa:Pastype;
                rr:objectMap [
                        rml:reference "Pastype"
                ]
        ];
        rr:predicateObjectMap [
                rr:predicate pa:uitgegevendoor;
                rr:objectMap [
                        rml:reference "Pasuitgever"
                ]
        ];
        rr:predicateObjectMap [
                rr:predicate pa:pashouder_postcode;
                rr:objectMap [
                        rml:reference "Pashouder postcode"
                ]
        ];
        rr:predicateObjectMap [
                rr:predicate pa:pashouder_plaats;
                rr:objectMap [
                        rml:reference "Pashouder plaats"
                ]
        ];
        rr:predicateObjectMap [
                rr:predicate ac:acceptant;
                rr:objectMap [
                        rr:parentTriplesMap <#AcceptantMapping>;
                ];
  ]

.
<#AcceptantMapping>
  rml:logicalSource [
                rml:source "/afstuderen/experimentParentSameCSV/input/transacties_modified.csv";
    rml:referenceFormulation ql:CSV
  ];
  rr:subjectMap [
                rr:template "http://transactie.example.com/{Acceptant}";
  ];
```

```
        rr:predicateObjectMap [
                rr:predicate ac:acceptant_postcode;
                rr:objectMap [
                        rml:reference "Acceptant postcode"
                ]
        ];
.
```

PoC – Kleinschalige mapping met meerdere bronnen

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix rml: <http://semweb.mmlab.be/ns/rml#> .
@prefix ql: <http://semweb.mmlab.be/ns/ql#> .
@prefix tr: <http://example.org/dc/elements/1.1/transactie#> .
@prefix pa: <http://example.org/dc/elements/1.1/pas#>.
@prefix ac: <http://example.org/dc/elements/1.1/acceptant#> .
@prefix pu: <http://example.org/dc/elements/1.1/pasuitgever#> .
@prefix zt: <http://example.org/dc/elements/1.1/> .

<#PasMapping>
  rml:logicalSource [
                rml:source "/afstuderen/experimentMultipleCSV/Input/transacties_modified.csv";
    rml:referenceFormulation ql:CSV
  ];
  rr:subjectMap [
                rr:template "http://transactie.example.com/{Pasnummer}";
                rr:class pa:Pasnummer
  ];
  rr:predicateObjectMap [
                rr:predicate pa:Pastype;
                rr:objectMap [
                        rml:reference "Pastype"
                ]
        ];
        rr:predicateObjectMap [
                rr:predicate pa:uitgegevendoor;
                rr:objectMap [
                        rml:reference "Pasuitgever"
                ]
        ];
        rr:predicateObjectMap [
                rr:predicate pa:pashouder_postcode;
                rr:objectMap [
                        rml:reference "Pashouder postcode"
                ]
        ];
        rr:predicateObjectMap [
                rr:predicate pa:pashouder_plaats;
                rr:objectMap [
                        rml:reference "Pashouder plaats"
                ]
        ];
        rr:predicateObjectMap [
                rr:predicate zt:datum;
                rr:objectMap [
                        rr:parentTriplesMap <#Weer> ;

                                rr:joinCondition
                                [

                                        rr:child "date";
                                        rr:parent "datetext";
                                ]

                ];
  ]
.
<#Weer>
  rml:logicalSource [
                rml:source "/afstuderen/experimentMultipleCSV/input/weer2.csv";
    rml:referenceFormulation ql:CSV
```

```
  ];
  rr:subjectMap [
                rr:template "http://transactie.example.com/weer/{datetext}";
                rr:class zt:datetext
  ];
        rr:predicateObjectMap [
                rr:predicate zt:regen;
                rr:objectMap [
                        rml:reference "Regen"
                ]
        ];
.
```

PoC – Grootschalige mapping met 1 bron

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix rml: <http://semweb.mmlab.be/ns/rml#> .
@prefix ql: <http://semweb.mmlab.be/ns/ql#> .
@prefix tr: <http://example.org/dc/elements/1.1/transactie#> .
@prefix pa: <http://example.org/dc/elements/1.1/pas#>.
@prefix ac: <http://example.org/dc/elements/1.1/acceptant#> .
@prefix pu: <http://example.org/dc/elements/1.1/pasuitgever#> .

<#PasMapping>
  rml:logicalSource [
                rml:source "/afstuderen/experimentLargeCSV/Input/ss10pusd.csv";
    rml:referenceFormulation ql:CSV
  ];
        rr:subjectMap [
                rr:template "http://transactie.example.com/{serialno}";
                rr:class ac:serialNo
  ];
  rr:predicateObjectMap [
                rr:predicate ac:RT;
                rr:objectMap [
                        rml:reference "RT"
                ]
        ];
        rr:predicateObjectMap [
                rr:predicate pa:SPORDER;
                rr:objectMap [
                        rml:reference "SPORDER"
                ]
        ];

.
```

Bijlage 13 Prototype Zeelandpas Case

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix rml: <http://semweb.mmlab.be/ns/rml#> .
@prefix ql: <http://semweb.mmlab.be/ns/ql#> .
@prefix tr: <http://example.org/dc/elements/1.1/transactie#> .
@prefix pa: <http://example.org/dc/elements/1.1/pas#>.
@prefix ac: <http://example.org/dc/elements/1.1/acceptant#> .
@prefix pu: <http://example.org/dc/elements/1.1/pasuitgever#> .
@prefix zt: <http://example.org/dc/elements/1.1/> .
@prefix wd: <http://example.org/dc/elements/1.1/weerdag#> .
@prefix pl: <http://example.org/dc/elements/1.1/plaats#> .
@prefix zp: <http://zeelandpas.nl/> .
@prefix zt: <http://example.org/dc/elements/1.1/> .

<#Transactie>
  rml:logicalSource [
                rml:source "/afstuderen/experimentZeelandPas/input/transacties.csv";
    rml:referenceFormulation ql:CSV
  ];
  rr:subjectMap [
                rr:template "http://transactie.example.com/{id}";
                rr:class tr:id
```

```
  ];
  rr:predicateObjectMap [
                        rr:predicate tr:datum;
                        rr:objectMap [
                                rml:reference "Datum"
                        ]
            ];
            rr:predicateObjectMap [
                        rr:predicate tr:categorie;
                        rr:objectMap [
                                rml:reference "Categorie"
                        ]
            ];
            rr:predicateObjectMap [
                        rr:predicate tr:prestatiecode;
                        rr:objectMap [
                                rml:reference "Prestatiecode"
                        ]
            ];
            rr:predicateObjectMap [
                        rr:predicate ac:acceptant;
                        rr:objectMap [
                                rr:parentTriplesMap <#Acceptant> ;
                        ]
  ];
            rr:predicateObjectMap [
                        rr:predicate pu:pasuitgever;
                        rr:objectMap [
                                rr:parentTriplesMap <#Pasuitgever> ;
                        ]
  ];
            rr:predicateObjectMap [
                        rr:predicate pa:Pasnummer;
                        rr:objectMap [
                                rr:parentTriplesMap <#Pasnummer> ;
                        ]
  ];
            rr:predicateObjectMap [
                        rr:predicate tr:datum1;
                        rr:objectMap [
                                rr:parentTriplesMap <#Weer> ;
                                rr:joinCondition
                                [
                                        rr:child "datum1";
                                        rr:parent "datetext";
                                ]
                        ]
  ];
.
<#Acceptant>
  rml:logicalSource [
                        rml:source "/afstuderen/experimentZeelandPas/input/transacties.csv";
    rml:referenceFormulation ql:CSV
  ];
  rr:subjectMap [
                        rr:template "http://transactie.example.com/acceptant/{Acceptant}";
                        rr:class ac:acceptant
  ];
            rr:predicateObjectMap [
                        rr:predicate ac:acceptant_postcode;
                        rr:objectMap [
                                rml:reference "Acceptant postcode"
                        ]
            ];
            rr:predicateObjectMap [
                        rr:predicate ac:acceptant_plaats;
                        rr:objectMap [
                                rr:parentTriplesMap <#Plaats> ;
                                rr:joinCondition
                                [
```

```
                                        rr:child "Acceptant plaats";
                                        rr:parent "Plaats";
                                ]
                        ]
  ];
.
<#Pasuitgever>
  rml:logicalSource [
                        rml:source "/afstuderen/experimentZeelandPas/input/transacties.csv";
    rml:referenceFormulation ql:CSV
  ];
  rr:subjectMap [
                        rr:template "http://transactie.example.com/pasuitgever/{Pasuitgever code}";
                        rr:class pu:pasuitgever_code
  ];
        rr:predicateObjectMap [
                        rr:predicate pu:pasuitgever;
                        rr:objectMap [
                                rml:reference "Pasuitgever"
                        ]
        ];
        rr:predicateObjectMap [
                        rr:predicate pu:pasuitgever_postcode;
                        rr:objectMap [
                                rml:reference "Pasuitgever postcode"
                        ]
        ];
        rr:predicateObjectMap [
                        rr:predicate pu:pasuitgever_plaats;
                        rr:objectMap [
                                rr:parentTriplesMap <#Plaats> ;
                                rr:joinCondition
                                [
                                        rr:child "Pasuitgever plaats";
                                        rr:parent "Plaats";
                                ]
                        ]
  ];
.
<#Pasnummer>
  rml:logicalSource [
                        rml:source "/afstuderen/experimentZeelandPas/input/transacties.csv";
    rml:referenceFormulation ql:CSV
  ];
  rr:subjectMap [
                        rr:template "http://transactie.example.com/pasnummer/{Pasnummer}";
                        rr:class pa:Pasnummer
  ];
        rr:predicateObjectMap [
                        rr:predicate pa:Pastype;
                        rr:objectMap [
                                rml:reference "Pastype"
                        ]
        ];
        rr:predicateObjectMap [
                        rr:predicate pa:pashouder_postcode;
                        rr:objectMap [
                                rml:reference "Pashouder postcode"
                        ]
        ];
        rr:predicateObjectMap [
                        rr:predicate pa:uitgegevendoor;
                        rr:objectMap [
                                rml:reference "Pasuitgever"
                        ]
        ];
        rr:predicateObjectMap [
                        rr:predicate pa:pashouder_plaats;
                        rr:objectMap [
                                rr:parentTriplesMap <#Plaats> ;
```

```
                                        rr:joinCondition
                                        [
                                                rr:child "Pashouder plaats";
                                                rr:parent "Plaats";
                                        ]
                        ]
  ];
.
<#Plaats>
  rml:logicalSource [
                        rml:source "/afstuderen/experimentZeelandPas/input/gemeentes.csv";
    rml:referenceFormulation ql:CSV
  ];
  rr:subjectMap [
                        rr:template "http://transactie.example.com/plaats/{Plaats}";
                        rr:class pl:plaats
  ];
        rr:predicateObjectMap [
                        rr:predicate pl:beschrijving;
                        rr:objectMap [
                                rml:reference "Beschrijving"
                        ]
        ];
        rr:predicateObjectMap [
                        rr:predicate pl:postcode;
                        rr:objectMap [
                                rml:reference "Postcode"
                        ]
        ];
        rr:predicateObjectMap [
                        rr:predicate pl:afstand;
                        rr:objectMap [
                                rr:parentTriplesMap <#Afstand> ;
                                rr:joinCondition
                                [
                                        rr:child "Gemeente";
                                        rr:parent "gemeentea";
                                ]
                        ]
  ];
.
<#Afstand>
  rml:logicalSource [
                        rml:source "/afstuderen/experimentZeelandPas/input/afstanden.csv";
    rml:referenceFormulation ql:CSV
  ];
  rr:subjectMap [
                        rr:template "http://transactie.example.com/afstand/afstand{gemeentea}{gemeenteb}";
                        rr:class zp:afstand
  ];
        rr:predicateObjectMap [
                        rr:predicate zt:gemeentea;
                        rr:objectMap [
                                rml:reference "gemeentea"
                        ]
        ];
        rr:predicateObjectMap [
                        rr:predicate zt:gemeenteb;
                        rr:objectMap [
                                rml:reference "gemeenteb"
                        ]
        ];
        rr:predicateObjectMap [
                        rr:predicate zt:afstand;
                        rr:objectMap [
                                rml:reference "afstand"
                        ]
        ];
.
<#Weer>
```

```
rml:logicalSource [
                rml:source "/afstuderen/experimentZeelandPas/input/weer2.csv";
   rml:referenceFormulation ql:CSV
];
rr:subjectMap [
                rr:template "http://transactie.example.com/weer/{YYYYMMDD}";
                rr:class wd:YYYYMMDD
];
        rr:predicateObjectMap [
                rr:predicate wd:regen;
                rr:objectMap [
                        rml:reference "Regen"
                ]
        ];
        rr:predicateObjectMap [
                rr:predicate wd:day;
                rr:objectMap [
                        rml:reference "day"
                ]
        ];
        rr:predicateObjectMap [
                rr:predicate wd:month;
                rr:objectMap [
                        rml:reference "month"
                ]
        ];
        rr:predicateObjectMap [
                rr:predicate wd:year;
                rr:objectMap [
                        rml:reference "year"
                ]
        ];
        rr:predicateObjectMap [
                rr:predicate wd:RH;
                rr:objectMap [
                        rml:reference "RH"
                ]
        ];
        rr:predicateObjectMap [
                rr:predicate wd:RHX;
                rr:objectMap [
                        rml:reference "RHX"
                ]
        ];
        rr:predicateObjectMap [
                rr:predicate wd:EV24;
                rr:objectMap [
                        rml:reference "EV24"
                ]
        ];
        rr:predicateObjectMap [
                rr:predicate wd:VVN;
                rr:objectMap [
                        rml:reference "VVN"
                ]
        ];
        rr:predicateObjectMap [
                rr:predicate wd:VVNH;
                rr:objectMap [
                        rml:reference "VVNH"
                ]
        ];
        rr:predicateObjectMap [
                rr:predicate wd:VVX;
                rr:objectMap [
                        rml:reference "VVX"
                ]
        ];
        rr:predicateObjectMap [
                rr:predicate wd:VVXH;
```

```
                rr:objectMap [
                        rml:reference "VVXH"
                ]
        ];
        rr:predicateObjectMap [
                rr:predicate wd:RHXH;
                rr:objectMap [
                        rml:reference "RHXH"
                ]
        ];
        rr:predicateObjectMap [
                rr:predicate wd:NG;
                rr:objectMap [
                        rml:reference "NG"
                ]
        ];
        rr:predicateObjectMap [
                rr:predicate wd:DR;
                rr:objectMap [
                        rml:reference "DR"
                ]
        ];
.
```

Bijlage 14 Prototype TransactieCompare

```
@prefix zt:      <http://example.org/dc/elements/1.1/> .
@prefix :        <http://example.org/dc/trans/> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix rml: <http://semweb.mmlab.be/ns/rml#> .
@prefix ql: <http://semweb.mmlab.be/ns/ql#> .

<#Transactie>
  rml:logicalSource [
                rml:source "/afstuderen/experimentZeelandPas/input/transacties.csv";
    rml:referenceFormulation ql:CSV
  ];
  rr:subjectMap [
                rr:template "http://transactie.example.com/{id}";
                rr:class zt:id
  ];
  rr:predicateObjectMap [
                rr:predicate zt:id;
                rr:objectMap [
                        rml:reference "id"
                ]
        ];
  rr:predicateObjectMap [
                rr:predicate zt:datum;
                rr:objectMap [
                        rml:reference "Datum"
                ]
        ];
        rr:predicateObjectMap [
                rr:predicate zt:pasnummer;
                rr:objectMap [
                        rml:reference "Pasnummer" ;
                ]
  ];
        rr:predicateObjectMap [
                rr:predicate zt:pastype;
                rr:objectMap [
                        rml:reference "Pastype" ;
                ]
  ];
        rr:predicateObjectMap [
                rr:predicate zt:pasuitgever_code;
                rr:objectMap [
```

```
                        rml:reference "Pasuitgever code" ;
                ]
    ];
        rr:predicateObjectMap [
                rr:predicate zt:pasuitgever;
                rr:objectMap [
                        rml:reference "Pasuitgever" ;
                ]
    ];
        rr:predicateObjectMap [
                rr:predicate zt:pasuitgever_postcode;
                rr:objectMap [
                        rml:reference "Pasuitgever postcode" ;
                ]
    ];
        rr:predicateObjectMap [
                rr:predicate zt:pasuitgever_plaats;
                rr:objectMap [
                        rml:reference "Pasuitgever plaats" ;
                ]
    ];
        rr:predicateObjectMap [
                rr:predicate zt:pashouder_postcode;
                rr:objectMap [
                        rml:reference "Pashouder postcode" ;
                ]
    ];
        rr:predicateObjectMap [
                rr:predicate zt:pashouder_plaats;
                rr:objectMap [
                        rml:reference "Pashouder plaats" ;
                ]
    ];
        rr:predicateObjectMap [
                rr:predicate zt:pashouder_land;
                rr:objectMap [
                        rml:reference "Pashouder land" ;
                ]
    ];
        rr:predicateObjectMap [
                rr:predicate zt:acceptant;
                rr:objectMap [
                        rml:reference "Acceptant" ;
                ]
    ];
        rr:predicateObjectMap [
                rr:predicate zt:acceptant_postcode;
                rr:objectMap [
                        rml:reference "Acceptant postcode" ;
                ]
    ];
        rr:predicateObjectMap [
                rr:predicate zt:acceptant_plaats;
                rr:objectMap [
                        rml:reference "Acceptant plaats" ;
                ]
    ];
        rr:predicateObjectMap [
                rr:predicate zt:prestatiecode;
                rr:objectMap [
                        rml:reference "Prestatiecode" ;
                ]
    ];
.
```

Bijlage 15 JUnit tests

JUnit tests lijken sterk op elkaar, het enige wat echt verschilt, is het uit te voeren SPARQL query. Daarom is er voor gekozen om 1 JUnit test volledig weer te geven, en de overige testen tot SPARQL query te beperken. Vervolgens is er een globaal transactie vergelijking uitgevoerd.

DatumCompare

```
package jUnitTests;

import static org.junit.Assert.*;
import org.junit.Test;
import com.hp.hpl.jena.query.QueryExecution;
import com.hp.hpl.jena.query.QueryExecutionFactory;
import com.hp.hpl.jena.query.ResultSet;
import com.hp.hpl.jena.sparql.resultset.ResultSetCompare;

public class DatumCompare {

	/*
	 * QueryOld = RMLProcessor Data
	 * QueryNew = Comparison Data
	 */

	//Build Query Strings
	static String QueryOld = "prefix tr: <http://example.org/dc/elements/1.1/transactie#>"
			+ "SELECT DISTINCT(?o as ?datum)"
			+ "FROM <http://localhost:3131/old> "
			+ "WHERE {?s tr:datum ?o}"
			+ "ORDER BY ?o";

	static String QueryNew = "prefix zt:        <http://example.org/dc/elements/1.1/> "
			+ "SELECT DISTINCT(?o as ?datum)"
			+ "FROM <http://localhost:3131/new> "
			+ "WHERE {?s zt:datum ?o}"
			+ "ORDER BY ?o";

	private boolean checkPassed;

	@Test
	public void test() {
		//Build and run the queries
		QueryExecution qeo = QueryExecutionFactory.sparqlService(
				"http://localhost:3131/ds/query", QueryOld);
		ResultSet resultsOld = qeo.execSelect();

		QueryExecution qen = QueryExecutionFactory.sparqlService(
				"http://localhost:3131/ds/query", QueryNew);
		ResultSet resultsNew = qen.execSelect();

		//Compare the results of the queries
		checkPassed = ResultSetCompare.equalsByValueAndOrder(resultsOld, resultsNew);
		System.out.print(checkPassed);
		assertTrue(checkPassed);

		//Close the connection
		qeo.close();
		qen.close();


	}

}
```

IdCompare

```
	//Build Query Strings
	static String QueryOld = "prefix tr: <http://example.org/dc/elements/1.1/transactie#>"
			+ "SELECT DISTINCT(?o as ?id)"
			+ "FROM <http://localhost:3131/old> "
			+ "WHERE {?s tr:id ?o}"
			+ "ORDER BY ?o";
```

```
static String QueryNew = "prefix zt:       <http://example.org/dc/elements/1.1/> "
                         + "SELECT DISTINCT(?o as ?id)"
                         + "FROM <http://localhost:3131/new> "
                         + "WHERE {?s zt:id ?o}"
                         + "ORDER BY ?o";
```

PasnummerCompare

```
//Build Query Strings
static String QueryOld = "prefix pa: <http://example.org/dc/elements/1.1/pas#>"
                         + "prefix rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#> "
                         + "SELECT DISTINCT(?o as ?pasnummer)"
                         + "FROM <http://localhost:3131/old> "
                         + "WHERE {?s rdf:type pa:Pasnummer."
                         + "?s pa:Pasnummer ?o}"
                         + "ORDER BY ?o";

static String QueryNew = "prefix zt:       <http://example.org/dc/elements/1.1/> "
                         + "SELECT DISTINCT(?o as ?pasnummer)"
                         + "FROM <http://localhost:3131/new> "
                         + "WHERE {?s zt:pasnummer ?o}"
                         + "ORDER BY ?o";
```

PastypeCompare

```
//Build Query Strings
static String QueryOld = "prefix pa: <http://example.org/dc/elements/1.1/pas#>"
                         + "SELECT DISTINCT(?o as ?pastype)"
                         + "FROM <http://localhost:3131/old> "
                         + "WHERE {?s pa:Pastype ?o}"
                         + "ORDER BY ?o";

static String QueryNew = "prefix zt:       <http://example.org/dc/elements/1.1/> "
                         + "SELECT DISTINCT(?o as ?pastype)"
                         + "FROM <http://localhost:3131/new> "
                         + "WHERE {?s zt:pastype ?o}"
                         + "ORDER BY ?o";
```

PasuitgeverCompare

```
//Build Query Strings
static String QueryOld = "prefix pu:       <http://example.org/dc/elements/1.1/pasuitgever#>"
                         + "prefix rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#> "
                         + "SELECT DISTINCT(?o as ?pasuitgever)"
                         + "FROM <http://localhost:3131/old> "
                         + "WHERE {?s rdf:type pu:pasuitgever_code."
                         + "?s pu:pasuitgever ?o}"
                         + "ORDER BY ?o";

static String QueryNew = "prefix zt: <http://example.org/dc/elements/1.1/> "
                         + "SELECT DISTINCT(?o as ?pasuitgever)"
                         + "FROM <http://localhost:3131/new> "
                         + "WHERE {?s zt:pasuitgever ?o}"
                         + "ORDER BY ?o";
```

PasuitgeverPostcodeCompare

```
//Build Query Strings
static String QueryOld = "prefix pu: <http://example.org/dc/elements/1.1/pasuitgever#>  "
                         + "SELECT DISTINCT(?o as ?pasuitgever)"
                         + "FROM <http://localhost:3131/old> "
                         + "WHERE {?s pu:pasuitgever_postcode ?o}"
                         + "ORDER BY ?o";

static String QueryNew = "prefix zt: <http://example.org/dc/elements/1.1/> "
                         + "SELECT DISTINCT(?o as ?pasuitgever)"
                         + "FROM <http://localhost:3131/new> "
                         + "WHERE {?s zt:pasuitgever_postcode ?o}"
                         + "ORDER BY ?o";
```

PostcodeCompare

```
//Build Query Strings
static String QueryOld = "prefix pl: <http://example.org/dc/elements/1.1/plaats#>"
```

```
                + "SELECT DISTINCT(?o as ?postcode)"
                + "FROM <http://localhost:3131/old> "
                + "WHERE {?s pl:postcode ?o}"
                + "ORDER BY ?o";

        static String QueryNew = "prefix zt:       <http://example.org/dc/elements/1.1/> "
                + "SELECT DISTINCT(?o as ?postcode)"
                + "FROM <http://localhost:3131/new> "
                + "WHERE {?s zt:postcode ?o}"
                + "ORDER BY ?o";
```

VVXHCompare

```
        //Build Query Strings
        static String QueryOld = "prefix wd: <http://example.org/dc/elements/1.1/weerdag#>"
                + "SELECT (?o as ?VVXH)"
                + "FROM <http://localhost:3131/old> "
                + "WHERE {?s wd:VVXH ?o}"
                + "ORDER BY ?o";

        static String QueryNew = "prefix zt:       <http://example.org/dc/elements/1.1/> "
                + "SELECT (?o as ?id)"
                + "FROM <http://localhost:3131/new> "
                + "WHERE {?s zt:VVXH ?o}"
                + "ORDER BY ?o";
```

TransactieCompare

```
package jUnitTests;

import static org.junit.Assert.*;

import org.junit.Test;

import com.hp.hpl.jena.query.*;

public class TransactieCompare {

        /*
         * QueryOld = RMLProcessor Data
         * QueryNew = Comparison Data
         */

        //Build Query Strings
        static String QueryOldString = "prefix rr: <http://www.w3.org/ns/r2rml#>"
                + "prefix rml: <http://semweb.mmlab.be/ns/rml#> "
                + "prefix ql: <http://semweb.mmlab.be/ns/ql#> "
                + "prefix tr: <http://example.org/dc/elements/1.1/transactie#> "
                + "prefix pa: <http://example.org/dc/elements/1.1/pas#>"
                + "prefix ac: <http://example.org/dc/elements/1.1/acceptant#> "
                + "prefix pu: <http://example.org/dc/elements/1.1/pasuitgever#> "
                + "prefix zt: <http://example.org/dc/elements/1.1/> "
                + "prefix wd: <http://example.org/dc/elements/1.1/weerdag#> "
                + "prefix pl: <http://example.org/dc/elements/1.1/plaats#> "
                + "prefix zp: <http://zeelandpas.nl/> "
                + "prefix zt: <http://example.org/dc/elements/1.1/> "
                + "SELECT (?id1 as ?id) (?datum1 as ?datum) (?pasnummer1 as ?pasnummer) (?pastype1 as ?pastype)
(?pasuitgever_code1 as ?pasuitgever_code) (?pasuitgever1 as ?pasuitgever) (?pu_postcode as ?pasuitgever_postcode) (?puplaats as
?pasuitgever_plaats) (?ph_postcode as ?pashouder_postcode) (?ph_plaats as ?pashouder_plaats) (?ph_land as ?pashouder_land)
(?acceptant1 as ?acceptant) (?ac_postcode as ?acceptant_postcode) (?ac_plaats as ?acceptant_plaats) (?prescode as ?prestatie_code)"
                + "FROM <http://localhost:3131/old> "
                + "WHERE {"
                + "?s tr:id ?id1."
                + "?s tr:datum ?datum1."
                + "?s pa:Pasnummer ?pas."
                + "?pas pa:Pasnummer ?pasnummer1."
                + "?pas pa:Pastype ?pastype1."
                + "?s pu:pasuitgever ?pu."
                + "?pu pu:pasuitgever_code ?pasuitgever_code1."
                + "?pu pu:pasuitgever ?pasuitgever1."
                + "?pu pu:pasuitgever_postcode ?pu_postcode."
```

```
                + "?pu pu:pasuitgever_plaats ?puplaats1."
                + "?puplaats1 pl:plaats ?puplaats."
                + "?pas pa:pashouder_plaats ?ph_plaats."
                + "?pas pa:pashouder_postcode ?ph_postcode."
                + "?pas pa:pashouder_land ?ph_land."
                + "?s ac:acceptant ?ac."
                + "?ac ac:acceptant ?acceptant1."
                + "?ac ac:acceptant_postcode ?ac_postcode."
                + "?ac ac:acceptant_plaats ?acplaats."
                + "?acplaats pl:plaats ?ac_plaats."
                + "?s tr:prestatiecode ?prescode."
                + "}"
                + "ORDER BY ?id1";

        static String QueryNewString = "prefix zt:       <http://example.org/dc/elements/1.1/>"
                + "SELECT (?id1 as ?id) (?datum1 as ?datum) (?pasnummer1 as ?pasnummer) (?pastype1 as ?pastype)
(?pasuitgever_code1 as ?pasuitgever_code) (?pasuitgever1 as ?pasuitgever) (?pu_postcode as ?pasuitgever_postcode) (?pu_plaats as
?pasuitgever_plaats) (?ph_postcode as ?pashouder_postcode) (?ph_plaats as ?pashouder_plaats) (?ph_land as ?pashouder_land)
(?acceptant1 as ?acceptant) (?ac_postcode as ?acceptant_postcode) (?ac_plaats as ?acceptant_plaats) (?prescode as ?prestatie_code)"
                + "FROM <http://localhost:3131/new> "
                + "WHERE {"
                + "?s zt:id ?id1."
                + "?s zt:datum ?datum1."
                + "?s zt:pasnummer ?pasnummer1."
                + "?s zt:pastype ?pastype1."
                + "?s zt:pasuitgever_code ?pasuitgever_code1."
                + "?s zt:pasuitgever ?pasuitgever1."
                + "?s zt:pasuitgever_postcode ?pu_postcode."
                + "?s zt:pasuitgever_plaats ?pu_plaats."
                + "?s zt:pashouder_postcode ?ph_postcode."
                + "?s zt:pashouder_plaats ?ph_plaats."
                + "?s zt:Pashouder_land ?ph_land."
                + "?s zt:acceptant ?acceptant1."
                + "?s zt:acceptant_postcode ?ac_postcode."
                + "?s zt:acceptant_plaats ?ac_plaats."
                + "?s zt:prestatiecode ?prescode."
                + "}"
                + "ORDER BY ?id1";

        private long limit = 100;
        private int counterMatch;
        private int counterNoMatch;

        @Test
        public void test() {
                //Build and run the queries
                Query queryOld = QueryFactory.create(QueryOldString);
                queryOld.setLimit(limit);
                Query queryNew = QueryFactory.create(QueryNewString);
                queryNew.setLimit(limit);

                QueryExecution qeo = QueryExecutionFactory.sparqlService(
                                "http://localhost:3131/ds/query", queryOld);
                ResultSet resultsOld = qeo.execSelect();

                QueryExecution qen = QueryExecutionFactory.sparqlService(
                                "http://localhost:3131/ds/query", queryNew);
                ResultSet resultsNew = qen.execSelect();

                while (resultsOld.hasNext() && resultsNew.hasNext())
                {
                        String stringOld = resultsOld.next().toString();
                        String stringNew = resultsNew.next().toString();
                        if (stringOld.equals(stringNew))
                        {
                                counterMatch++;
                        }
                        else {
                                System.out.println(stringOld);
                                System.out.println(stringNew);
```

```
                                System.out.println("Not matching!");
                                counterNoMatch++;
                        }
                }
                System.out.println("Matching: " +counterMatch);
                System.out.println("Not Matching: " +counterNoMatch);



                assertTrue(counterNoMatch <= 5);

                //Close the connection
                qeo.close();
                qen.close();


        }
}
```

TransactieCompare2

```
package jUnitTests;

import static org.junit.Assert.*;

import org.junit.Test;

import com.hp.hpl.jena.query.*;

public class TransactieCompare2 {

        /*
         * QueryOld = RMLProcessor Data
         * QueryNew = Comparison Data
         */

        //Build Query Strings
        static String QueryOldString = "prefix zt:      <http://example.org/dc/elements/1.1/>"
                                + "SELECT (?id1 as ?id) (?datum1 as ?datum) (?pasnummer1 as ?pasnummer) (?pastype1 as ?pastype)
(?pasuitgever_code1 as ?pasuitgever_code) (?pasuitgever1 as ?pasuitgever) (?pu_postcode as ?pasuitgever_postcode) (?pu_plaats as
?pasuitgever_plaats) (?ph_postcode as ?pashouder_postcode) (?ph_plaats as ?pashouder_plaats) (?ph_land as ?pashouder_land)
(?acceptant1 as ?acceptant) (?ac_postcode as ?acceptant_postcode) (?ac_plaats as ?acceptant_plaats) (?prescode as ?prestatie_code)"
                                + "FROM <http://localhost:3131/trans> "
                                + "WHERE {"
                                + "?s zt:id ?id1."
                                + "?s zt:datum ?datum1."
                                + "?s zt:pasnummer ?pasnummer1."
                                + "?s zt:pastype ?pastype1."
                                + "?s zt:pasuitgever_code ?pasuitgever_code1."
                                + "?s zt:pasuitgever ?pasuitgever1."
                                + "?s zt:pasuitgever_postcode ?pu_postcode."
                                + "?s zt:pasuitgever_plaats ?pu_plaats."
                                + "?s zt:pashouder_postcode ?ph_postcode."
                                + "?s zt:pashouder_plaats ?ph_plaats."
                                + "?s zt:pashouder_land ?ph_land."
                                + "?s zt:acceptant ?acceptant1."
                                + "?s zt:acceptant_postcode ?ac_postcode."
                                + "?s zt:acceptant_plaats ?ac_plaats."
                                + "?s zt:prestatiecode ?prescode."
                                + "}"
                                + "ORDER BY ?id1";

        static String QueryNewString = "prefix zt:      <http://example.org/dc/elements/1.1/>"
                                + "SELECT (?id1 as ?id) (?datum1 as ?datum) (?pasnummer1 as ?pasnummer) (?pastype1 as ?pastype)
(?pasuitgever_code1 as ?pasuitgever_code) (?pasuitgever1 as ?pasuitgever) (?pu_postcode as ?pasuitgever_postcode) (?pu_plaats as
?pasuitgever_plaats) (?ph_postcode as ?pashouder_postcode) (?ph_plaats as ?pashouder_plaats) (?ph_land as ?pashouder_land)
(?acceptant1 as ?acceptant) (?ac_postcode as ?acceptant_postcode) (?ac_plaats as ?acceptant_plaats) (?prescode as ?prestatie_code)"
```

```
                + "FROM <http://localhost:3131/new> "
                + "WHERE {"
                + "?s zt:id ?id1."
                + "?s zt:datum ?datum1."
                + "?s zt:pasnummer ?pasnummer1."
                + "?s zt:pastype ?pastype1."
                + "?s zt:pasuitgever_code ?pasuitgever_code1."
                + "?s zt:pasuitgever ?pasuitgever1."
                + "?s zt:pasuitgever_postcode ?pu_postcode."
                + "?s zt:pasuitgever_plaats ?pu_plaats."
                + "?s zt:pashouder_postcode ?ph_postcode."
                + "?s zt:pashouder_plaats ?ph_plaats."
                + "?s zt:Pashouder_land ?ph_land."
                + "?s zt:acceptant ?acceptant1."
                + "?s zt:acceptant_postcode ?ac_postcode."
                + "?s zt:acceptant_plaats ?ac_plaats."
                + "?s zt:prestatiecode ?prescode."
                + "}"
                + "ORDER BY ?id1";

private long limit = 100000;
private float counterTotal;
private float counterMatch;
private float counterNoMatch;

@Test
public void test() {
        //Build and run the queries
        Query queryOld = QueryFactory.create(QueryOldString);
        queryOld.setLimit(limit);
        Query queryNew = QueryFactory.create(QueryNewString);
        queryNew.setLimit(limit);

        QueryExecution qeo = QueryExecutionFactory.sparqlService(
                        "http://localhost:3131/ds/query", queryOld);
        ResultSet resultsOld = qeo.execSelect();

        QueryExecution qen = QueryExecutionFactory.sparqlService(
                        "http://localhost:3131/ds/query", queryNew);
        ResultSet resultsNew = qen.execSelect();

        while (resultsOld.hasNext() && resultsNew.hasNext())
        {
                String stringOld = resultsOld.next().toString();
                String stringNew = resultsNew.next().toString();
                counterTotal++;
                if (stringOld.equals(stringNew))
                {
                        counterMatch++;
                }
                else {
                        System.out.println(stringOld);
                        System.out.println(stringNew);
                        System.out.println("Not matching!");
                        counterNoMatch++;
                }
        }
        System.out.println("Total: " + counterTotal);
        System.out.println("Matching: " +counterMatch);
        System.out.println("Not Matching: " +counterNoMatch);
        System.out.println("Percent not matching: % " + counterNoMatch/counterTotal*100);

        //Compare if less or equal to 5%
        assertTrue(counterNoMatch/counterTotal*100 <= 5);

        //Close the connection
        qeo.close();
        qen.close();
```

```
        }
}
```

Bijlage 16 Requirements – MoSCoW Ingevuld

ID	Prioriteit	Eis	Status	Notes
REQ_001	Must	Framework moet data bestanden kunnen converteren naar RDF	Done	RML mapping
REQ_002	Should	Framework zou een optie moeten bieden om meerdrere bestanden achter elkaar unattended te converteren	Done	Batch files kunnen een aantal conversies achter elkaar laten uitvoeren
REQ_003	Should	Framework zou moeten geconverteerde bestanden naar een server sturen	To do	Handmatig
REQ_004	Should	Framework zou moeten geconverteerde bestanden in een RDF database laden	To do	Handmatig
REQ_005	Should	Framework zou moeten geconverteerde bestanden als backup bijhouden	Done	TTL file
REQ_006	Must	Framework moet een optie bieden om templates te koppelen met bestanden	Done	RML Mapping
REQ_007	Should	Framework zou grote bestanden (tot 500 MB) moeten ondersteunen	Done	RML wel, OR heeft minimaal 5GB RAM nodig
REQ_008	Could	Framework zou kunnen de conversie per row kunnen uitvoeren, om geheugen te besparen	Done	Gebeurt per row lijkt het
REQ_009	Must	Templates moeten ervoor zorgen dat de data op een uniforme wijze geformateerd wordt	Done	RML - kan ook velden "opbouwen" / OR voor transformaties
REQ_010	Won't	Framework zal niet in de achtergrond werken en zonder menselijke input starten/converteren	Removed	Het kan wel, maar het hoeft niet - batch files kunnen prima in de achtergrond runnen of handmatig opgestart worden
REQ_011	Should	Framework zou moeten een log-bestand genereren	To do	Command line debug info
REQ_012	Should	Framework zou het volgende logging niveaus moeten kennen: error, warning, info	To do	Command line debug info
REQ_013	Must	Framework moet een optie bieden om text velden te splitsen in afzonderlijke velden	Done	OR maakt dat mogelijk
REQ_014	Should	Framework zou moeten door een persoon met minimale ICT ervaring onderhoudbaar moeten zijn	Done	Batch files kunnen geheel automatisch uitgevoerd worden. Bestanden verplatsen kan met een handleiding
REQ_015	Should	Framework zou flexibel genoeg moeten zijn om in de toekomst ook andere data bestanden te ondersteunen	To do	RML processor kan altijd uitgewisseld/uitgebreidt worden met een ander conversie tool. Misschien verstandig om GUI te maken, en de tools "verborgen" te houden?
REQ_016	Should	Het zou mogelijk moeten zijn om template op te slaan om deze te hergebruiken	Done	RML en OR gebruiken templates dat hergebruikt kunnen worden

REQ_001 Framework moet data bestanden kunnen converteren naar RDF
Gebruikmakend van RML Mappings (zie Bijlage 12 en Bijlage 13) was het mogelijk om data bestanden (.csv) te converteren naar RDF (.ttl).

REQ_002 Framework zou een optie moeten bieden om meerdere bestanden achter elkaar unattended te converteren
Dat is mogelijk op twee manieren, allereerst kan er een batch file gemaakt worden, dat meerdere mappings achter elkaar uitvoert. Tweede optie is om een mapping te maken, die meerdere bestanden als invoer gebruikt, zoals Kleinschalige mapping met meerdere bronnen in Bijlage 12.

REQ_003 Framework zou moeten geconverteerde bestanden naar een server sturen
Dat gebeurt nu handmatig, bijvoorbeeld via control panel van Fuseki. Gezien het tijdbeperking en lage prioriteit is deze requirement niet uitgewerkt.

REQ_004 Framework zou moeten geconverteerde bestanden in een RDF database laden
Ook dat gebeurt nu handmatig. De control panel van Fuseki voert het versturen en inladen van een bestand achter elkaar.

REQ_005 Framework zou moeten geconverteerde bestanden als backup bijhouden
Het resultaat van de RML Processor is een .ttl bestand. Deze kan als backup gebruikt worden, en ingeladen in een triplestore.

REQ_006 Framework moet een optie bieden om templates te koppelen met bestanden
Een RML Mapping bevat een invoer sectie, daarin moet een pad naar een bestand gedefinieerd worden.

REQ_007 Framework zou grote bestanden (tot 500 MB) moeten ondersteunen
Zoals het uit de test blijkt, kan RML Processor goed omgaan met grote bestanden. OpenRefine kan met bestanden van ~500MB omgaan, mits er voldoende RAM geheugen aanwezig is. Voor details, zie 3.3.1.4.1.

REQ_008 Framework zou kunnen de conversie per row kunnen uitvoeren, om geheugen te besparen
Op basis van de resultaten uit 3.3.1.4.1, lijkt het alsof de RML Processor het bestand per row doorloopt. Het kleine verbruik van RAM geheugen bij zeer grote bestanden is vrij nihil, wat deze theorie bevestigt.

REQ_009 Templates moeten ervoor zorgen dat de data op een uniforme wijze geformatteerd wordt
De RML Processor voert zijn acties op basis van een template, voor een voorbeeld zie Bijlage 12 of Bijlage 13. OpenRefine maakt het mogelijk om alle uitgevoerde bewerkingen als JSON bestand op te slaan, om deze vervolgens in andere projecten toe te passen, voor voorbeeld zie Bijlage 18.

REQ_010 Framework zal niet in de achtergrond werken en zonder menselijke input starten/converteren
Deze wens bleek in het vervolg van het onderzoek niet meer te kloppen. Zo wou een van de stakeholders dat het wel mogelijk is. Uiteindelijk kan een RML Mapping uitgevoerd worden als een batch file. Het is dus mogelijk om deze handmatig, of automatisch te laten uitvoeren.

REQ_011 Framework zou moeten een log-bestand genereren
In verband met tijdgebrek is deze optie niet gerealiseerd. Wel wordt er een log gegenereerd in de command prompt. Het uitbreiden van deze functionaliteit naar een log-bestand zou dus mogelijk moeten zijn.

REQ_012 Framework zou het volgende logging niveaus moeten kennen: error, warning, info
In verband met tijdgebrek is deze optie niet gerealiseerd. Er is maar 1 logging niveau, dat alle conversies weergeeft, met op het einde informatie over aantal triples en het benodigde tijd.

REQ_013 Framework moet een optie bieden om tekst velden te splitsen in afzonderlijke velden
OpenRefine maakt het mogelijk om kolommen te splitsen op separators of lengte. Om een voorbeeld te zien, zie Bijlage 18.

REQ_014 Framework zou moeten door een persoon met minimale ICT ervaring onderhoudbaar moeten zijn
OpenRefine en RML Processor kunnen door een persoon met minimale ICT ervaring onderhoudt worden. In de usecases voor dit project wordt er vanuit gegaan van twee actors: expert en niet-expert. Het onderhoud, d.w.z. het uitvoeren van al ontworpen transformaties, kan door een niet-expert uitgevoerd worden. Om een voorbeeld hiervan te zien, zie Bijlage 6 en Bijlage 18.

REQ_015 Framework zou flexibel genoeg moeten zijn om in de toekomst ook andere data bestanden te ondersteunen
OpenRefine en RML Processor ondersteunen veel data formaten. In geval dat er behoefte is om deze functionaliteit uit te breiden, kan elk van deze elementen vervangen worden. Zolang het resultaat van een transformatie een .csv, JSON, XML of RDB bestand is, kam RML Processor mee omgaan. Dat wil zeggen dat OpenRefine vervangen kan worden, met bijvoorbeeld een ETL proces, zolang het output van deze formaat is. RML Processor kan ook vervangen worden met een ander elementen. Er is echter geen onderzoek uitgevoerd om te bewijzen of dat klopt, dus is het onmogelijk om te zeggen of er aan deze requirement is voldaan.

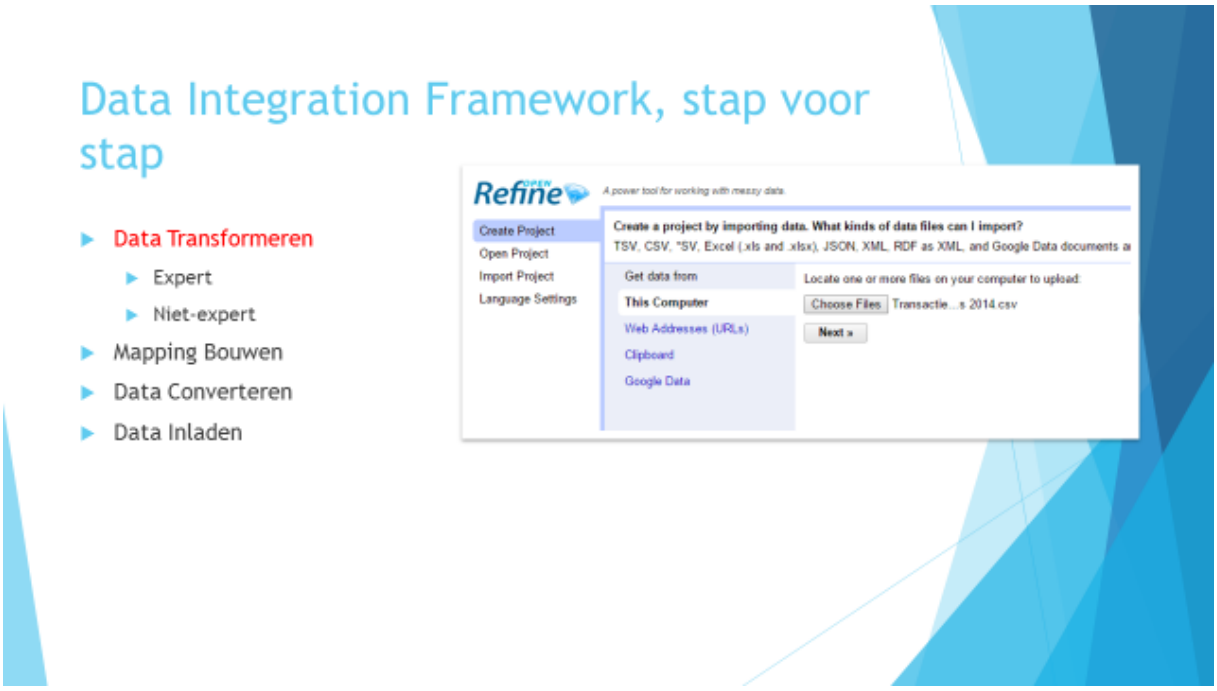
REQ_016 Het zou mogelijk moeten zijn om template op te slaan om deze te hergebruiken
Een RML Mapping en een OpenRefine JSON extract kunnen opgeslagen en hergebruikt worden. Voor voorbeelden zie Bijlage 12, Bijlage 13 en Bijlage 18.

Bijlage 17 Zoekplan

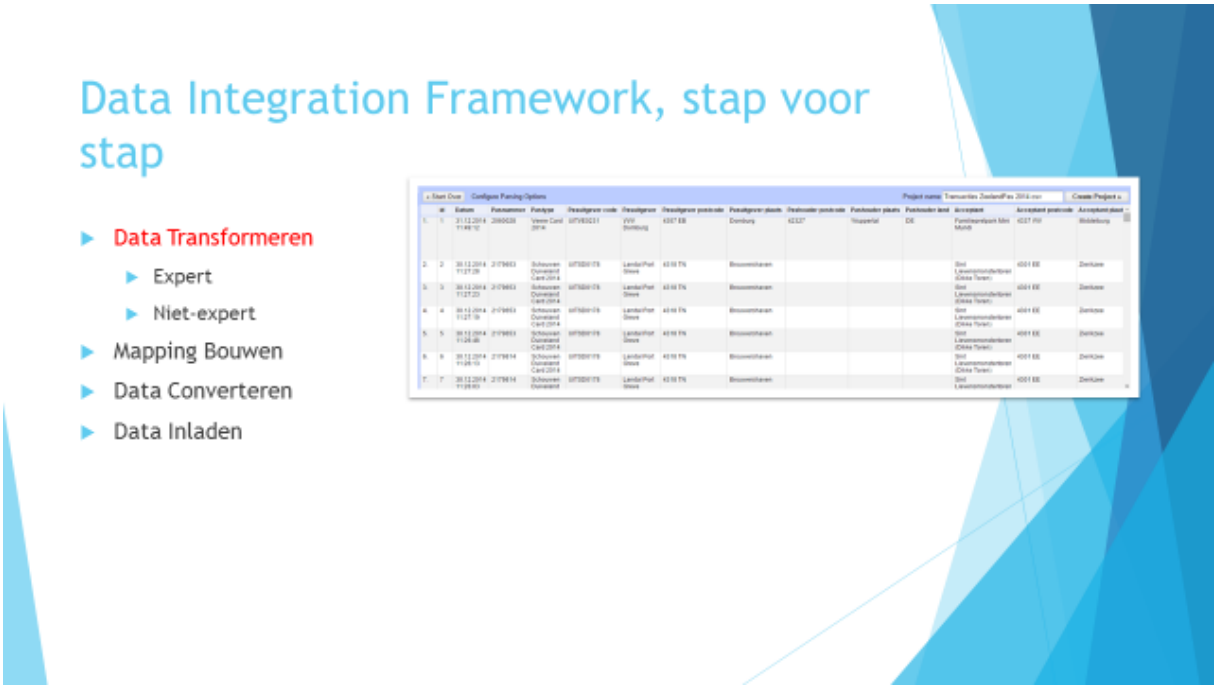
Wat wil ik weten?	Bron	Gebruikte zoekterm	Resultaat
Wat zijn op dit moment opties om naar RDF te converteren	Google	convert to rdf	http://www.w3.org/wiki/ConverterToRdf
Wat is google refine?	Google	Google refine	https://code.google.com/p/google-refine/
Wat is OpenRefine?	Google	OpenRefine	http://openrefine.org/
Hoe kan OpenRefine RDF ondersteunen?	Google	OpenRefine RDF	http://refine.deri.ie/
Welk bestandstypes worden door OpenRefine ondersteund?	Google	OpenRefine supported file	https://books.google.nl/books?id=_WG1AAAAQBAJ&printsec=frontcover&hl=nl
Wat houdt RDF skeleton in?	Google	RDF skeleton	http://refine.deri.ie/rdfExport
Welk tools zijn er beschikbaar op gebied van RDF data integratie?	Google	RDF data integration	http://www.isi.edu/integration/karma/
		triple store data integration	http://ldif.wbsg.de/
Wat is er beschikbaar op gebied van integratie van heterogene data	Google	heterogeneous data integration rdf	http://events.linkeddata.org/ldow2014/papers/ldow2014_paper_01.pdf
Wat houdt R2RML in?	Google	R2RML	http://www.w3.org/TR/r2rml/
Wat houdt RML in?	Google	rml rdf	http://semweb.mmlab.be/rml/spec.html
Wat houdt SKOS in?	Google	SKOS w3c	http://www.w3.org/TR/skos-reference/
Welk onderzoeken zijn al op gebied van ontology based data integration gedaan?	Google Scholar	Ontology based data integration	http://www.isif.org/fusion/proceedings/fusion07CD/Fusion07/pdfs/Fusion2007_1318.pdf
Wat is er bekend op gebied van syntactic heterogeneity?	Google	syntactic heterogeneity	https://books.google.nl/books?id=x4e2lVV0u9gC&pg=PA262&lpg=PA262&dq=syntactic+heterogeneity&source=bl&ots=_tg8jlmh2c&sig=A9hVodsT06gwQIrU28303hVAHJ0&hl=nl&sa=X&ei=ipgSVbXJJYG7UeK1gOgD&ved=0CDsQ6AEwAg#v=onepage&q=syntactic%20heterogeneity&f=false
Wat is er bekend op gebied van structural heterogeneity?	Google	structural heterogeneity data integration	https://books.google.nl/books?id=oWmp10vBI7cC&pg=PA442&lpg=PA442&dq=structural+heterogeneity+data+integration&source=bl&ots=moDGlUcEKI&sig=ZxB9QQB3gb05n7HEqV3dCLkK_kQ&hl=nl&sa=X&ei=36YSVbS3E4HTU5WVgcAH&ved=0CC4Q6AEwAjgK#v=onepage&q=structural%20heterogeneity%20data%20integration&f=false
Wat is er bekend op gebied van semantic heterogeneity?	Google scholar	semantic heterogeneity data integration	http://dl.acm.org/citation.cfm?id=263668
	Google	Managing Semantic Heterogeneity in Databases	http://groups.lis.illinois.edu/amag/langevgroup/localpapers/hull-managing-semantic-heterogeneity-pods-1997.pdf
Waar kan ik een grote CSV dataset krijgen?	Google	large csv dataset	http://lemire.me/blog/archives/2012/03/27/publicly-available-large-data-sets-for-database-research/

Bijlage 18 Data Integration Framework, handleiding

Data Transformeren



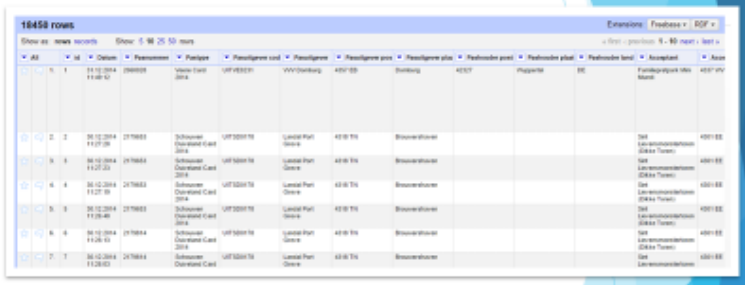
Allereerst moet er een project aangemaakt worden. Elk bestand dat getransformeerd moet worden, moet een eigen project hebben.



Zodra een bestand is gekozen en ingeladen, wordt er een overzicht gegenereerd van de aanwezige data. Als deze klopt, moet er vervolgens een naam ingevoerd worden, voordat het project opgeslagen kan worden.

Data Integration Framework, stap voor stap

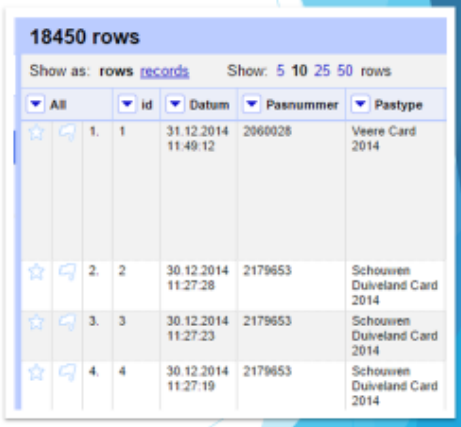
- Data Transformeren
  - Expert
  - Niet-expert
- Mapping Bouwen
- Data Converteren
- Data Inladen



Zodra een project is opgeslagen, wordt het werkruimte gepresenteerd aan de gebruiker. Hierin kunnen transformaties uitgevoerd worden op de geselecteerde bestand.

Data Integration Framework, stap voor stap

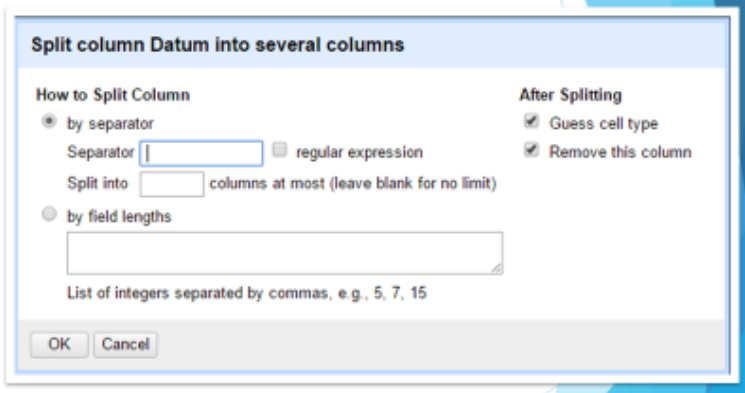
- Data Transformeren
  - Expert
  - Niet-expert
- Mapping Bouwen
- Data Converteren
- Data Inladen



Om de werking van OpenRefine te presenteren, wordt er een potentieel dilemma voorgesteld. Het databestand bevat “Datum” veld, dat een samenstelling is van datum en tijdstip. Een expert zou graag willen dat deze opgesplitst wordt in twee afzonderlijke kolommen. Om dat te bereiken moet de expert op de driehoekige knop in de kop van kolom klikken, en edit column -> Split into several columns te selecteren.

Data Integration Framework, stap voor stap

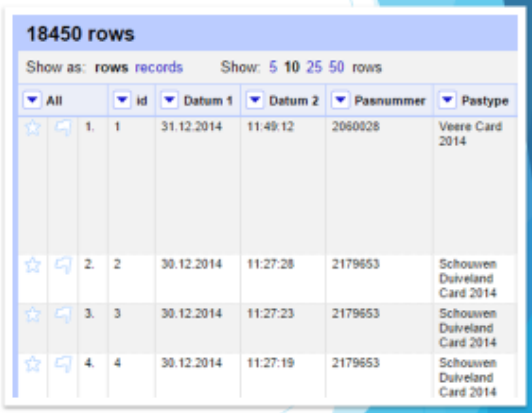
- Data Transformeren
  - Expert
  - Niet-expert
- Mapping Bouwen
- Data Converteren
- Data Inladen



De datum en tijdstip is gesepareerd door een spatie. Daarom is er in separator veld een spatie ingevoerd. Het is ook mogelijk om kolommen op andere criteria te splitsen, maar in dit geval is spatie als separator voldoende.

Data Integration Framework, stap voor stap

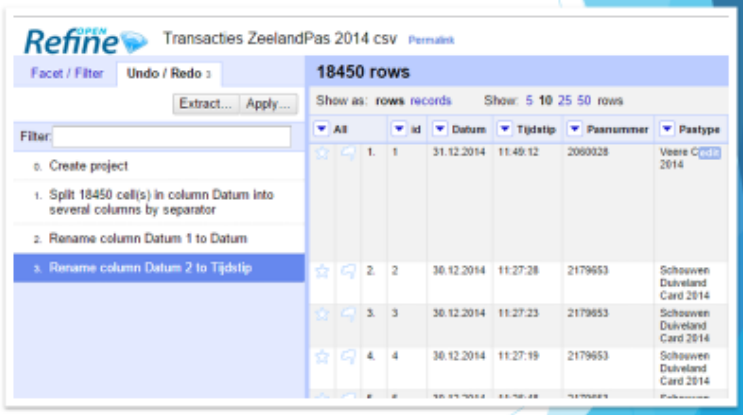
- Data Transformeren
  - Expert
  - Niet-expert
- Mapping Bouwen
- Data Converteren
- Data Inladen



De kolom is gesplitst in twee kolommen – Datum 1 en Datum 2. Dat is echter onoverzichtelijk, en dus kan het verstandig zijn om de kolom naam aan te passen.

Data Integration Framework, stap voor stap

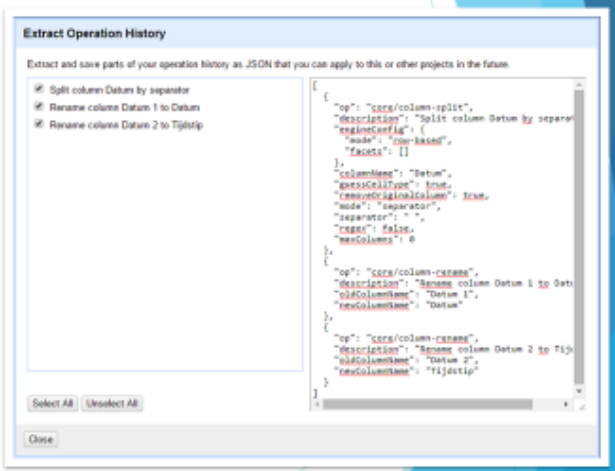
- Data Transformeren
  - Expert
  - Niet-expert
- Mapping Bouwen
- Data Converteren
- Data Inladen



Alle gedane bewerkingen worden bijgehouden. Zoals het hierboven te zien is, is er een project aangemaakt, een kolom is gesplitst in meerdere kolommen, en zijn beide resulterende kolommen van een ander naam voorzien. De gebruiker kan op elk willekeurige moment, terug gaan op basis van deze bewerkingen. Ook is het mogelijk om deze bewerkingen te extrageren door middel van het knop “Extract”.

Data Integration Framework, stap voor stap

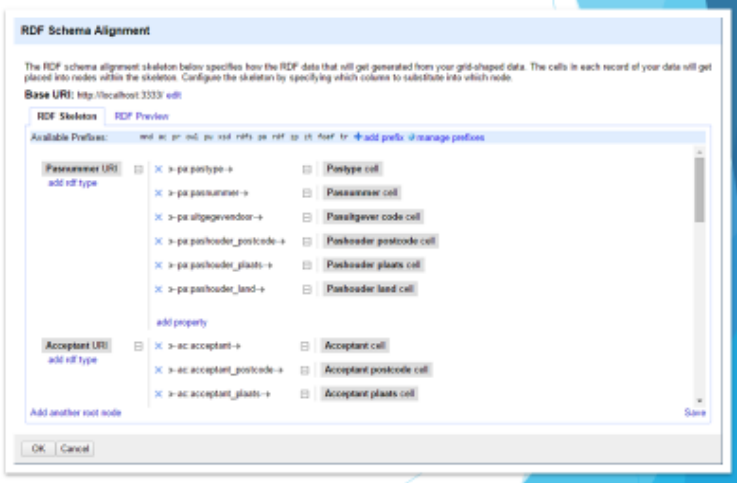
- Data Transformeren
  - Expert
  - Niet-expert
- Mapping Bouwen
- Data Converteren
- Data Inladen



Hierin is een voorbeeld van een extract van de bewerkingen. Deze heeft een vorm van een JSON bestand. Gebruikmakend van checkboxes is het mogelijk om alleen deel van de bewerkingen te selecteren. Het geëxtraheerde JSON bestand kan later gebruikt worden om diezelfde bewerkingen op een ander bestand uit te voeren, mits deze hetzelfde datastructuur heeft.

Data Integration Framework, stap voor stap

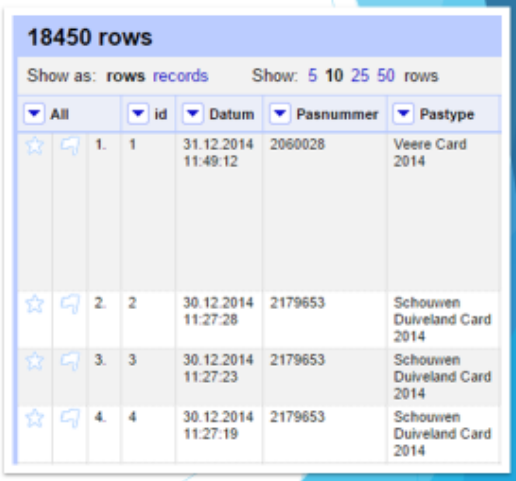
- Data Transformeren
  - Expert
  - Niet-expert
- Mapping Bouwen
- Data Converteren
- Data Inladen



Ook is het mogelijk om een RDF-skeleton aan te maken. Dat maakt het mogelijk om direct vanuit OpenRefine een .ttl bestand te genereren.

Data Integration Framework, stap voor stap

- Data Transformeren
  - Expert
  - Niet-expert
- Mapping Bouwen
- Data Converteren
- Data Inladen

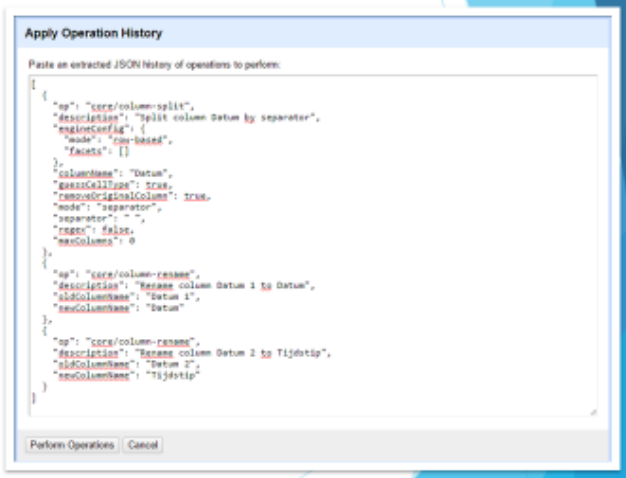


All		id	Datum	Pasnummer	Pastype
1.	1	31.12.2014 11:49:12	2060028	Veere Card 2014
2.	2	30.12.2014 11:27:28	2179653	Schouwen Duiveland Card 2014
3.	3	30.12.2014 11:27:23	2179653	Schouwen Duiveland Card 2014
4.	4	30.12.2014 11:27:19	2179653	Schouwen Duiveland Card 2014

Data integration framework is ook bedoeld om gebruikt te worden door niet-experts. Wat gebeurt er wanneer een niet-expert met diezelfde dilemma geconfronteerd wordt?

Data Integration Framework, stap voor stap

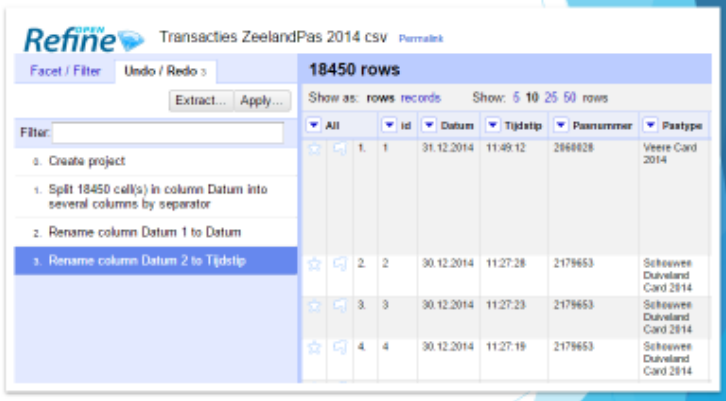
- Data Transformeren
  - Expert
  - Niet-expert
- Mapping Bouwen
- Data Converteren
- Data Inladen



Wanneer een set bewerkingen uitgevoerd worden door een expert, kunnen deze geëxtraheerd worden. De geëxtraheerd bewerking, in de vorm van een JSON bestand, kan vervolgens door een niet-expert gebruikt worden, om diezelfde bewerkingen uit te voeren. Om dat te doen, moet de gebruiker op Apply knop drukken, en de inhoud van JSON bestand er in plakken.

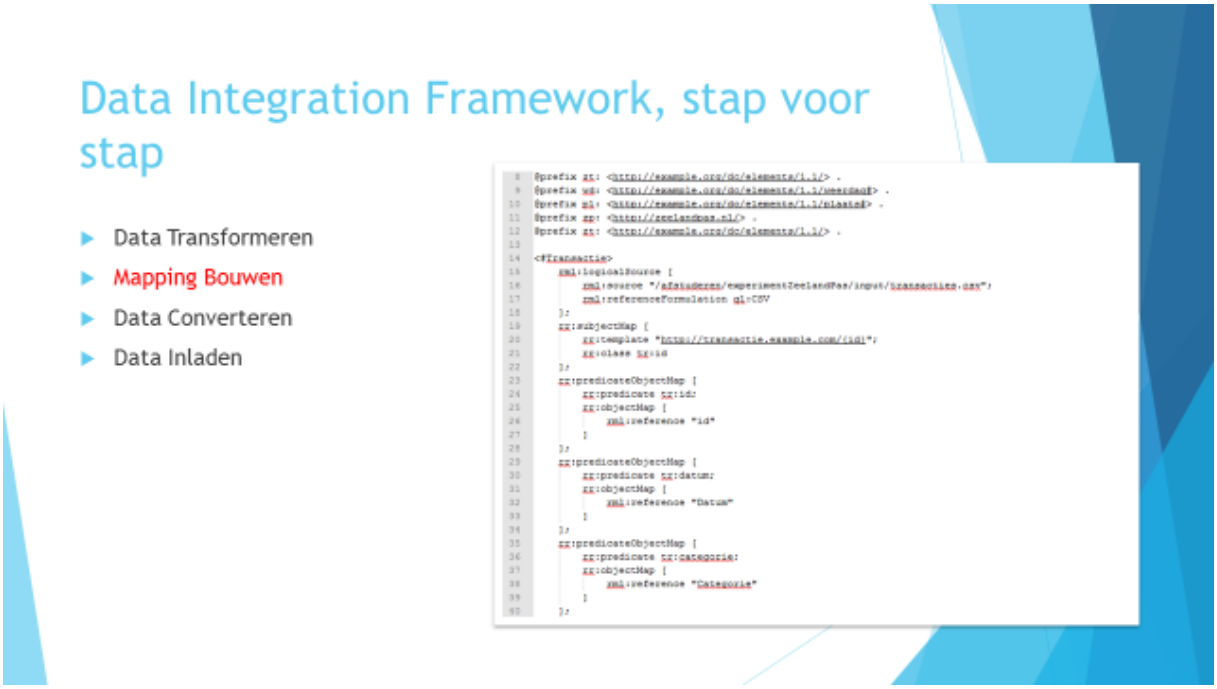
Data Integration Framework, stap voor stap

- Data Transformeren
  - Expert
  - Niet-expert
- Mapping Bouwen
- Data Converteren
- Data Inladen



Het resultaat is dan precies hetzelfde als die van een expert.

Mapping Bouwen



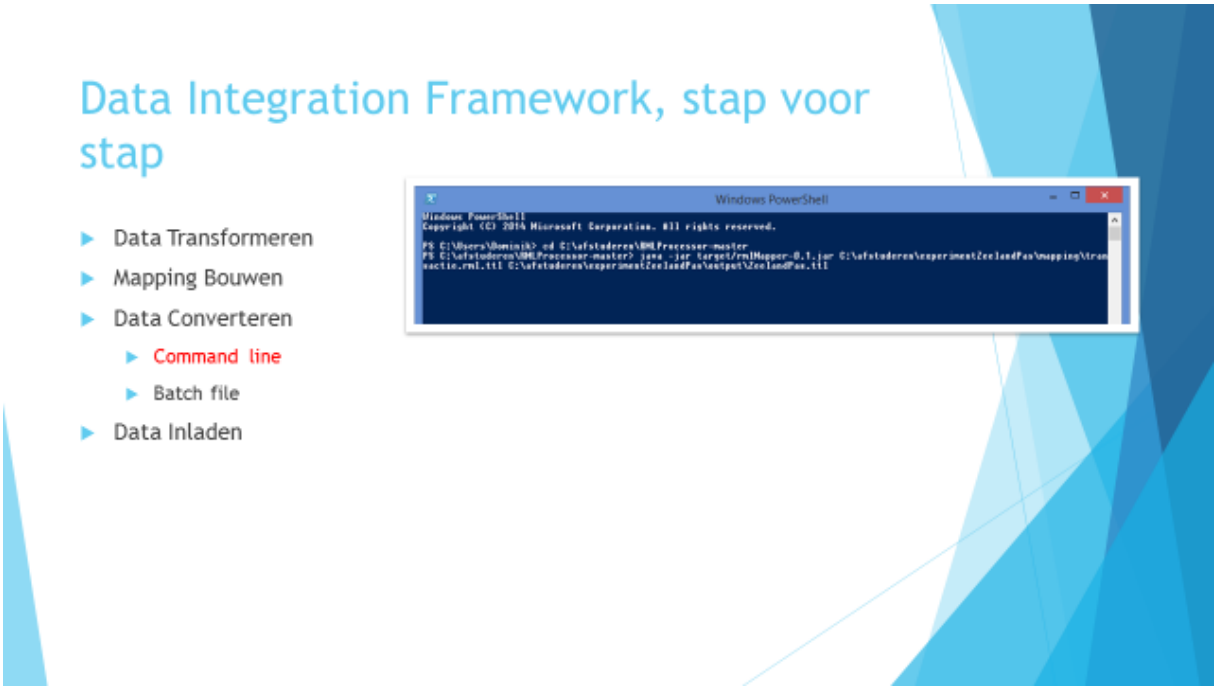
Een RML mapping heeft een voor van een .tll bestand. Zo'n mapping bestaat uit prefixes declaratie en een of meerdere triples maps. Triples map bestaat uit logical source, subject map en een over meerdere predicate-object maps.

Prefix declaratie wordt gebruikt om prefixes te kunnen gebruiken. Elk gebruikte prefix moet ook in prefix declaratie aanwezig zijn. Elk prefix declaratie begint met een "@" teken en eindigt met een punt.

Triples map definieert hoe een mapping uitgevoerd moet worden. Allereerst wordt er een logical source aangewezen. Dat is het directe pad naar het bestand die geconverteerd moet worden, en het type van het bestand (in dit geval: CSV)

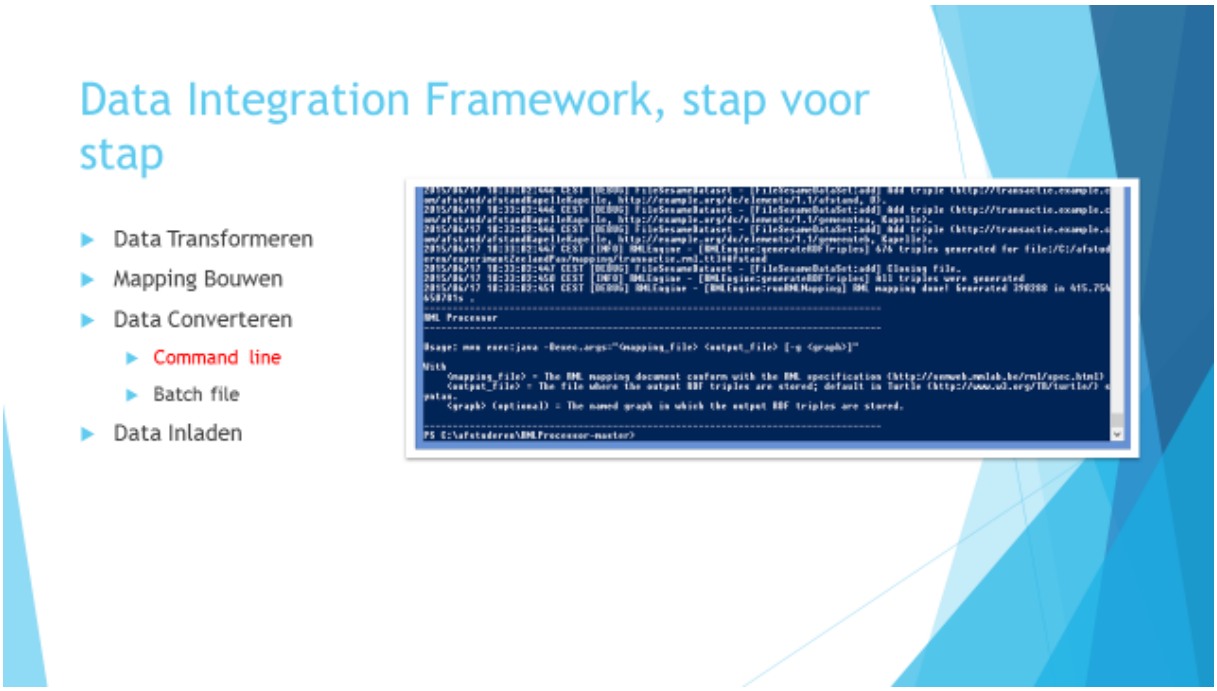
Subject map beschrijft hoe URI gegenereerd moet worden. In dit voorbeeld gebruiken we "http://transactie.example.com/(id)". Dat wil zeggen dat elk triple een URI krijgt, dat met "http://transactie.example.com" begint, en eindigt met een specifieke id. Deze id moet dan ook in het logical source te vinden zijn.

Predicate-object maps beschrijven de relatie tussen de URI en de onderliggende attributen. In dit voorbeeld wordt elk transactie voorzien van een id, datum en categorie. Elk van deze attributen krijgt een predicate (bijvoorbeeld tr:id of tr:datum). Vervolgens wordt deze predicate gebruikt om de koppeling te makken tussen de URI en de onderliggende object, waaraan de rml:reference naar refereert.

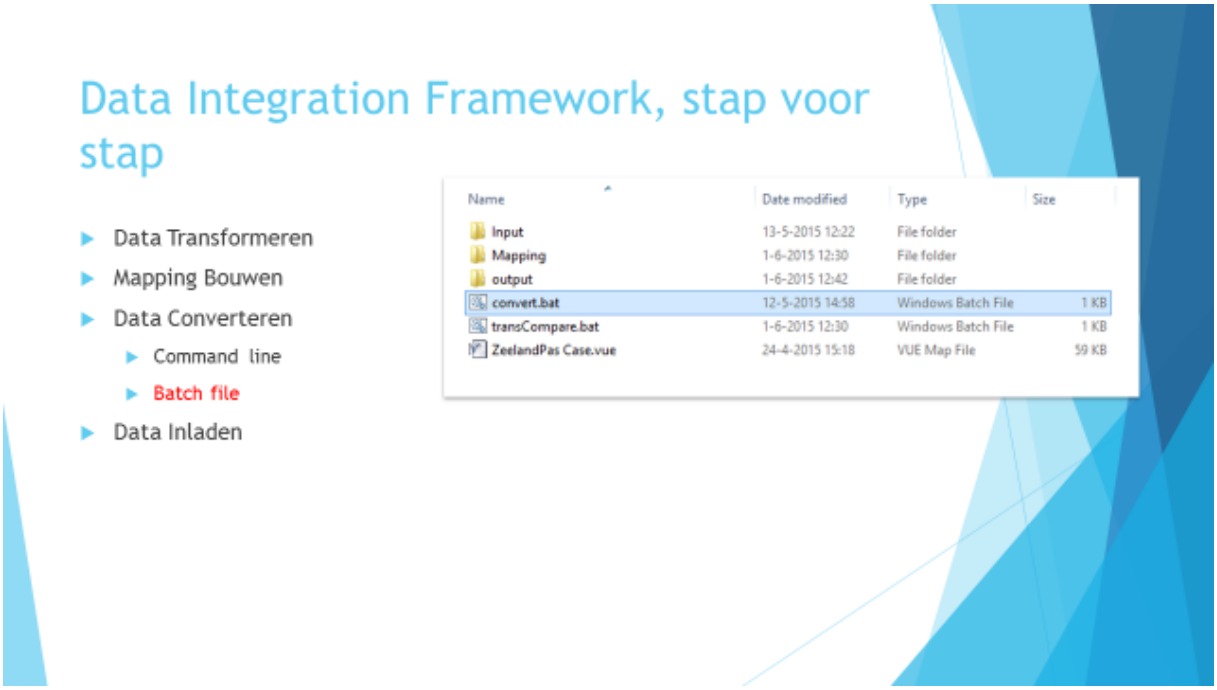


Om data te converteren moet er een commando aangeroepen worden, om de conversie uit te voeren. In dit voorbeeld wordt deze commando via Windows Powershell uitgevoerd, maar een ander command line tool is ook een optie.

Allereerst moet er naar de locatie van de rml-processor genavigeerd worden. Vervolgens moet er een commando aangeroepen worden met de volgende gegevens: “java –jar target/rmlMapper-0.1.jar” + locatie van de mapping + gewenste locatie van de output (inclusief de gewenste naam van het bestand).



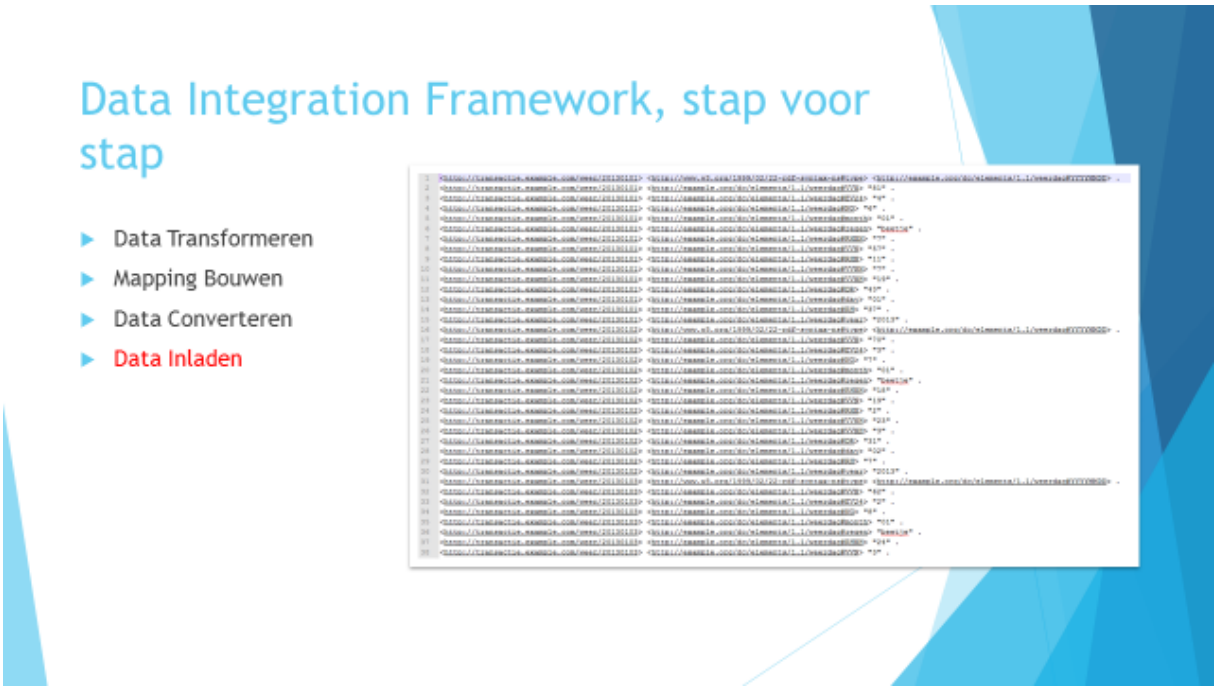
Nadat het commando wordt gegeven, wordt het conversie uitgevoerd. Dit proces kan afhankelijk van de grootte van gebruikte dataset en complexiteit van mapping wat tijd kosten. Op het einde van conversie wordt het aantal gegenereerd triples en de benodigde tijd weergegeven.



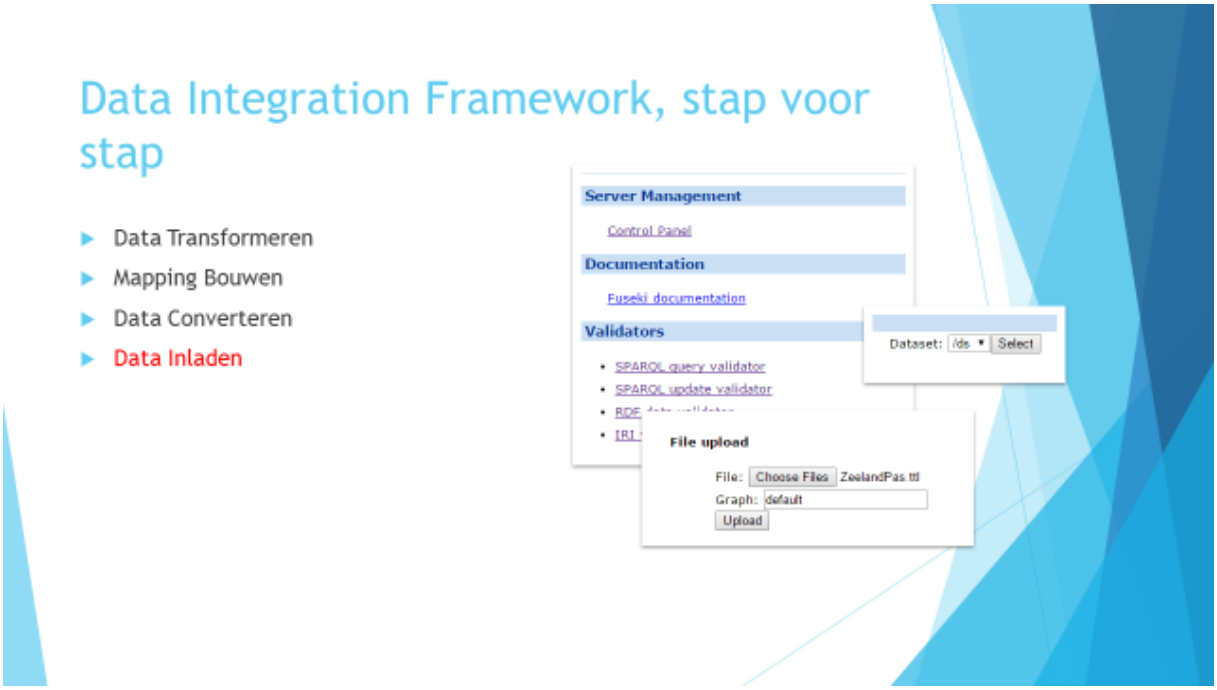
Gezien dat de Data Integration Framework ook door niet-experts uitgevoerd moet kunnen worden, is er onderzocht hoe dit proces versimpeld kan worden. Een van te voren ontworpen mappen-structuur in combinatie met een batchbestand dat de conversie uitvoert, leek zeer gebruikers vriendelijk.

In dit geval hoeft de gebruiker de gebruikte dataset in de input map plaatsen. Vervolgens, moet het convert.bat bestand uitgevoerd worden. Het resulterend data wordt in de output map geplaatst.

Data Inladen

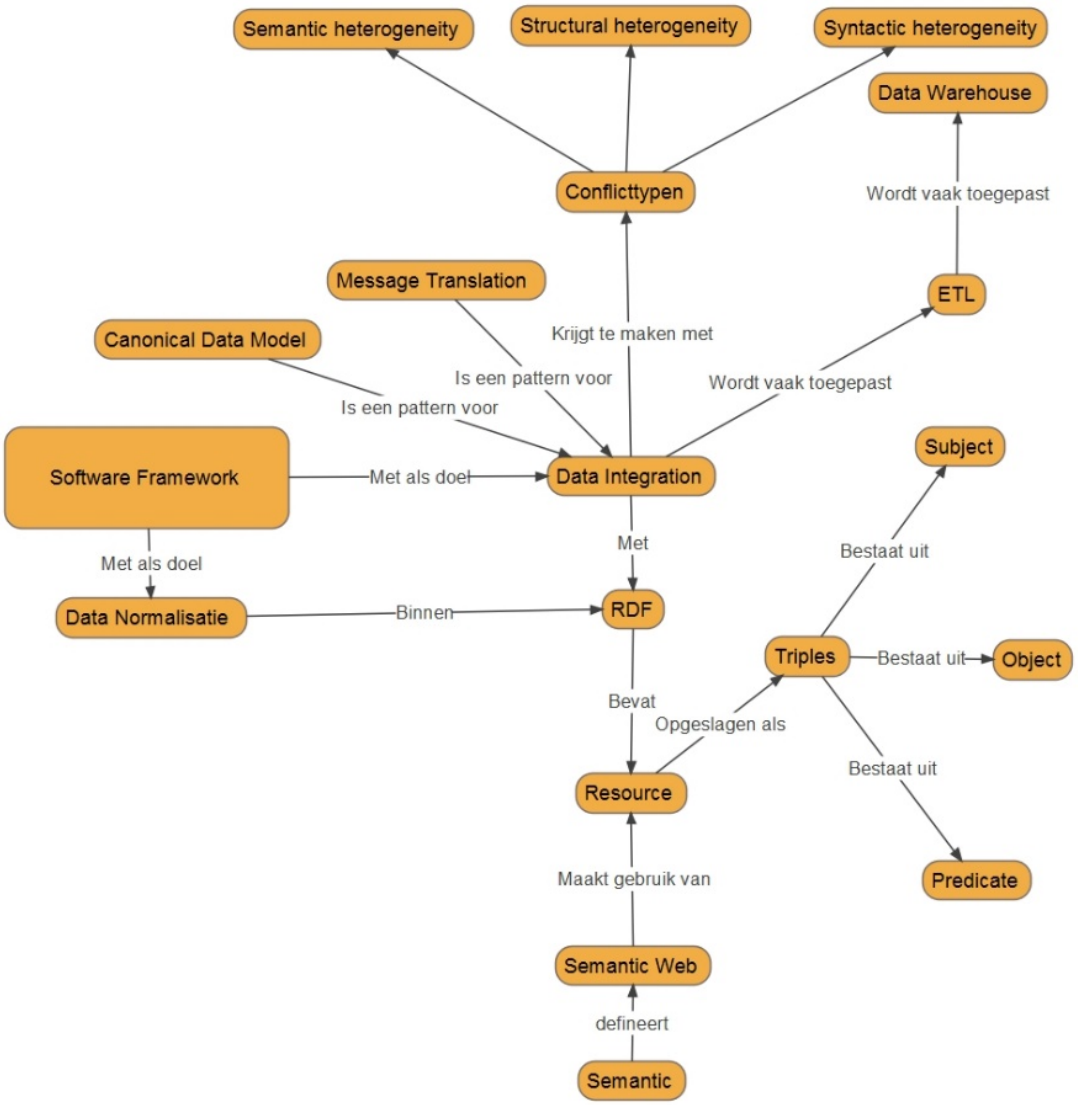


Het uitvoeren van conversie levert een .ttl bestand op. Deze kan bijvoorbeeld zo eruit zien.



Om de data in fuseki in te laden, kan er gebruik gemaakt worden van control panel. Om zo'n .ttl bestand up te loaden moet de gebruiker naar Control Panel navigeren, een dataset selecteren, en vervolgens naar beneden te scrollen. Daarin moet een graph geselecteerd worden (in dit geval default). Ook moet het eerder gegenereerd .ttl geselecteerd worden. Nadat deze gegevens gecontroleerd zijn, klikt de gebruiker op upload knop.

Bijlage 19 Samenhang Theoretisch Kader



Figuur 6