Datatranslatie met MT

Bachelorscriptie Informatica

9 juni 2008

Martijn Ermers

0116823

Radboud Universiteit Nijmegen

Inhoudsopgave

Overzicht gebruikte figuren

1 Introductie

We leven in een tijdperk met een overvloed aan digitale informatie. Er zijn veel uiteenlopende formaten waarin de informatie beschikbaar is. Denk aan databases, spreadsheets en dergelijke. Nu er steeds meer toepassingen beschikbaar worden welke gebruik maken van allerlei verschillende formaten van informatiebronnen, wordt het steeds belangrijker om informatiebronnen te kunnen transformeren van de beschikbare formaten naar de gewenste formaten. Dit kan ook een transformatie zijn van een structuur naar een andere binnen hetzelfde formaat (bijvoorbeeld een Database). Er zijn verschillende deelgebieden bij het transformeren van data.

Zo heb je het transformeren van een taal naar de andere. Hierbij blijft de informatie onaangetast, maar de overhead verandert. Hoewel deze transformaties uitermate zinvol kunnen zijn hebben deze nauwelijks academisch gehalte, want alles wordt een op een vertaald naar het gewenste formaat. De structuur wordt dus niet wezenlijk veranderd. Voor meer informatie over deze transformaties, ook wel data conversie genoemd, verwijs ik naar de papers [1] [2] en [3] . Het transformeren tussen verschillende talen wordt niet verder behandeld in deze scriptie.

Een ander deelgebied binnen het transformeren is het omzetten van één datastructuur in de andere. Hierbij verandert de structuur van de data wat wenselijk kan zijn voor bepaalde toepassingen. Hierbij is er wel sprake van academisch gehalte omdat hierbij intelligent bepaald moet worden wat de gewenste doelrepresentatie is en hiernaast het nog zo moet worden ingericht dat de data automatisch omgezet kan worden in deze representatie. Dit is het deelgebied waaraan ik de aandacht schenk binnen deze scriptie. De relevantie van dit soort transformaties definieer ik in hoofdstuk 2 .

Naast het redeneren over de relevantie van transformaties behandel ik ook in welke mate MT hiervoor geschikt is. Deze taal profileert zich als Modeltransformatietaal. Dit wil kortweg zeggen dat er vanuit een bronmodel een transformatie gemaakt kan worden in MT welke het doelmodel oplevert. Om te kunnen bepalen of MT geschikt is voor de gewenste transformaties zal ik een eenvoudig voorbeeld beschouwen welke toch geen triviale transformatie is. Dit voorbeeld is terug te vinden in hoofdstukken 5 en 6 . In Hoofdstuk 5 definieer ik de modeltransformatie onafhankelijk van de transformatietaal. Deze definitie gebruik ik om op terug te vallen bij de implementatie in MT in hoofdstuk 6 Verder gebruik ik dit hoofdstuk om de constructen van MT te verhelderen.

Verder geef ik in hoofdstuk 4 een beknopte uitleg van de voor MT relevante onderdelen van Converge en in hoofdstuk 7 zal ik de tegengekomen problemen verder toelichten.

2 Waarom modeltransformaties?

In de introductie heb ik al kort toegelicht waarom modeltransformaties zinvol zijn, maar de relevantie van modeltransformaties verdient meer uitleg welke ik in dit hoofdstuk zal geven. Eerst zal ik aangeven waarom er niet één beste database representatie is, daarna zal ik een voorbeeld geven wat benadrukt wanneer dit het geval kan zijn.

2.1. Waarom verschillende databases?

Er zijn verschillende databaserepresentaties die exact dezelfde informatie bevatten met dezelfde voorwaarden en dezelfde onderlinge relaties.

2.1.1. Conceptuele datamodel

Alle representaties hebben dan hetzelfde conceptuele datamodel. Dit model geeft precies aan welke informatie er is en op welke manier deze aan elkaar verbonden is. Bovendien kunnen er restricties op zitten zodat instanties van een feit (of verzameling feiten) uniek zijn. Ook kan er afgedwongen worden dat wanneer er informatie beschikbaar is van een feit dat die moet voorkomen in een ander feit. Als je bijvoorbeeld de feiten vaders en zonen hebt. Dan kun je de restricties opleggen dat iedere zoon slechts één vader heeft. Verder kun je aangeven dat iedere zoon een vader heeft. Je kunt meer lezen over conceptuele datamodellen in Van Bommel[4]

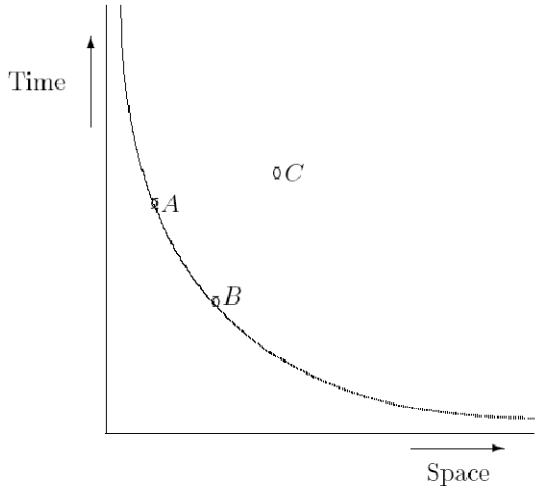
2.1.2. Factoren

Met dit conceptuele datamodel staat er vast waaraan een implementatie (database) moet voldoen. In alle niet triviale gevallen heb je een groot aantal keuzes van valide implementaties. Het kiezen van een geschikte representatie hangt af van de twee hoofdfactoren tijd en ruimte. Met tijd bedoel ik de snelheid waarmee informatie uit de database gelezen kan worden. Met ruimte bedoel ik de fysieke ruimte dat de database inneemt. Bijvoorbeeld op een harde schijf. De tijd kan verbeterd worden door de database zodanig in te richten dat veelgebruikte combinaties van informatie samen in een tabel te plaatsen.

De tijd wordt juist verslechterd wanneer de benodigde informatie indirect te verkrijgen is. Bijvoorbeeld doordat de sleutel(s) waarmee de benodigde informatie aan elkaar hangt op zichzelf geen benodigde informatie is.

De ruimte wordt verbeterd wanneer er minder redundantie bestaat. Dit is in het algemeen wanneer de tabellen kleiner worden (en dus meer tabellen). Een database met een tabel per feittype van het conceptuele datamodel bevat geen redundantie.

In het algemeen wordt de ruimte slechter wanneer de tijd beter wordt en visa versa. Ook zijn er voorbeelden te bedenken waarbij zowel de tijd als de ruimte slechter wordt, deze oplossingen dienen vermeden te worden. In onderstaande figuur wordt mooi geillustreerd dat je een afweging moet maken tussen tijd en ruimte. Verder zie je dat er ook zinloze opties bestaan zoals C. Deze oplossing is zowel slechter in tijd als in ruimte dan A en B. Alle opties op de kromme kunnen de gewenste oplossing zijn, al geldt in het algemeen dat er weinig toepassingen zijn waarbij een enorme toename van tijd een minimale ruimtebesparing kan verantwoorden. Zo ook visa versa, vaker zal men ergens gaan zitten waar er nog een redelijke kromming zit in de figuur.



Figuur 1 – Visualisatie Time/Space Tradeoff, uit van Bommel [4]

Er is nog een belangrijke factor welke samenhangt met ruimte, namelijk onderhoud. Hoe slechter de ruimte, hoe meer dubbele informatie aanwezig is. Dit is van belang bij het updaten van informatie. Wanneer er een gegeven gewijzigd wordt zal er meer veranderd moeten worden wanneer deze informatie op meerdere plekken staat. De benodigde tijd voor permutaties zal dus toenemen gelijk met de ruimte. Voor uitgebreidere informatie hierover verwijs ik naar Van Bommel[4] .

2.2. Voorbeeld

Een bedrijf heeft een database met klantgegevens, productgegevens, inkoopgegevens, verkoopgegevens. Uiteindelijk zijn alle gegevens aan elkaar te koppelen, al dan niet via een directe relatie.

Het bedrijf bestaat uit verschillende afdelingen welke allemaal op een zo snel mogelijke manier willen werken. Verder is er een IT afdeling welke voor de opslag moet zorgen.

De afdeling onderzoek wil graag dat de database zo is opgebouwd dat het makkelijk te zien is welke technieken gebruikt worden in veelverkochte producten en dus ook welke technieken relatief slecht zijn. De afdeling marketing wil kunnen zien wat voor soort koper een klant is en wil met deze informatie kunnen bepalen wat de gepaste informatie naar die klant toe moet. Voor de afdeling inkoop is het van belang welke producten in welke mate verkocht worden en wat daarvoor ingekocht moet worden. Tenslotte wil de IT afdeling dat de benodigde ruimte niet te groot wordt zodat er niet buitensporig geïnvesteerd hoeft te worden in hardware.

Kortom willen de verschillende afdelingen eigenlijk allemaal een andere database representatie hebben. Het kan wijs zijn iedere afdeling een eigen database representatie te geven. Hierdoor kan iedere afdeling op een efficiënte manier gebruik maken van de informatie. De IT afdeling kan kan een eenvoudige niet redundante representatie erop na houden om alle data up-to-date te houden. Wanneer er wijzigingen zijn kunnen deze aangepast worden in deze database, waarna de verschillende databases voor de verschillende afdelingen weer hieruit gegenereerd kunnen worden.

2.3. Bijwerken

Wanneer een database bijgewerkt wordt moet je altijd op de hoede zijn. Want wanneer informatie dubbel (redundant) aanwezig is in een representatie zul je deze op alle plekken moeten aanpassen. Ook hierbij kan datatransformatie een oplossing bieden. Wanneer je namelijk teruggaat naar een representatie waarbij geen redundantie mogelijk is kun je zonder zorgen over inconsistentie de informatie aanpassen. Na het aanpassen kun je de data weer transformeren naar de gewenste representatie. Een beknopt voorbeeld hiervan is te zien in 0

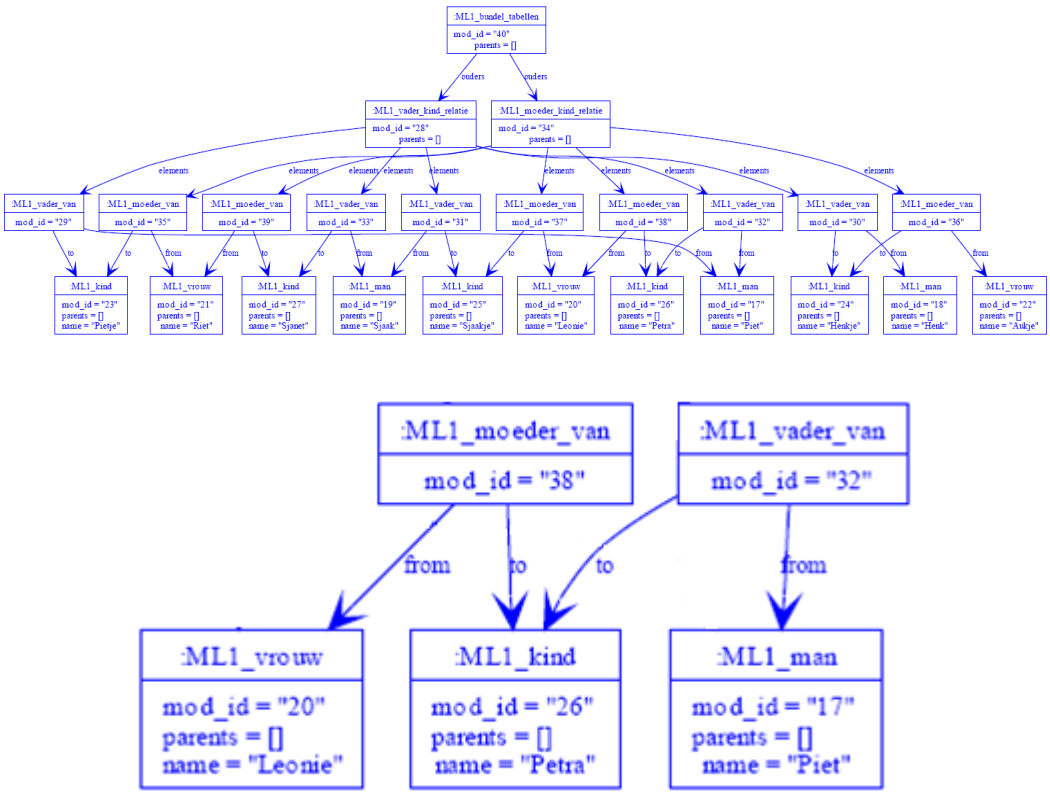
Merk op dat het tranformeren van de hele dataset in sommige gevallen wellicht overbodig is. In dit geval kun je inbouwen dat er door de computer intelligent bepaald wordt wat er getransformeerd dient te worden.

Verder zijn er ook dingen te verzinnen als checks op consistentie bij het transformeren van een redundante database naar een niet redundante database. Wanneer je bijvoorbeeld in een database op twee plekken hebt opgeslagen wat de naam is van een medewerker. Je deze naam aanpast op slechts één plek. Dan noem je deze database inconsistent.

2.3.1. Voorbeeld transformatie

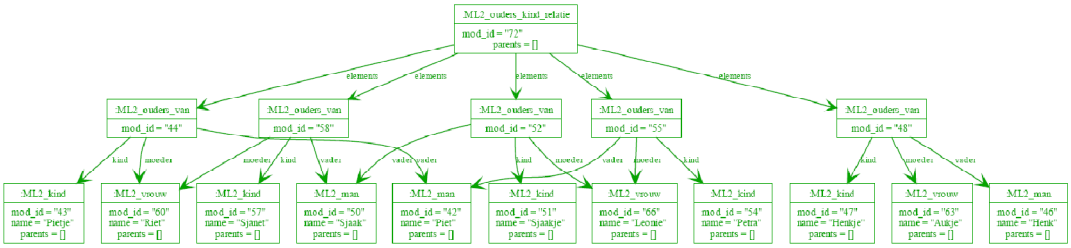
Dit voorbeeld is slechts bedoeld om te laten zien hoe men een transformatie kan gebruiken om aanpassingen van dubbele informatie gegarandeerd overal kan veranderen. De eenvoud van dit voorbeeld impliceert niet dat de toepasbaarheid beperkt is. Het doel van dit voorbeeld is ook niet om te beschrijven hoe een transformatie binnen MT werkt, wel is MT gebruikt om het voorbeeld te beschrijven.

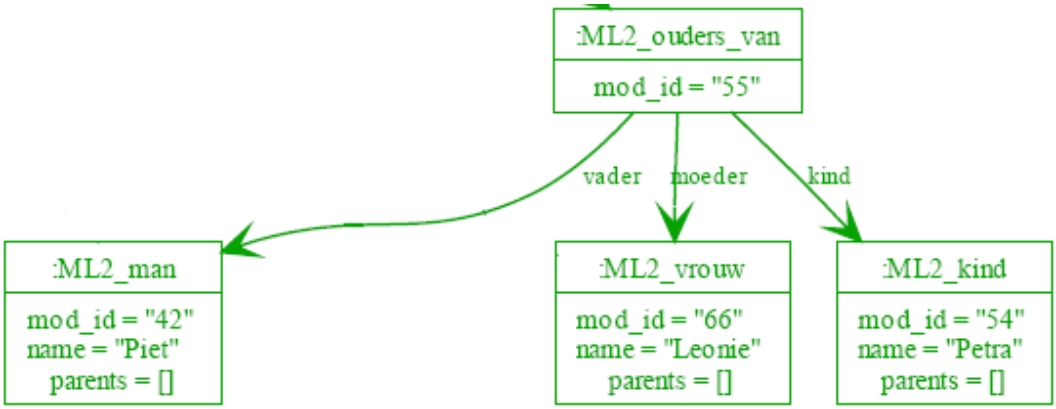
Het voorbeeld gaat uit van twee tabellen. De eerste tabel bevat namen van Vaders met de naam van hun Kind daarbij. De tweede tabel bevat namen van Moeders met de naam van hun Kind daarbij. In onderstaande schema staat een voorbeeld van deze tabellen. Merk op dat ik in deze en komende figuren de totale generatie van MT weergeef met daaronder een relevante selectie in groter formaat. Deze relevante selectie is ook ontdaan van irrelevante informatie voor dit voorbeeld.



Figuur 2 – Bronpopulatie onbewerkt, met uitvergroting.

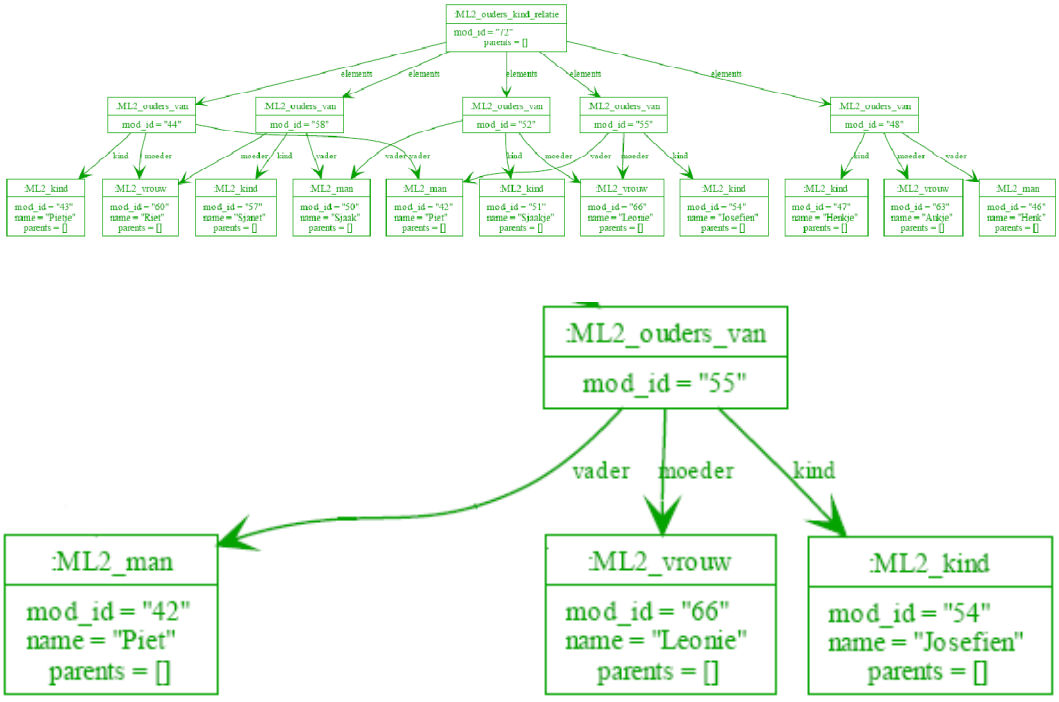
Als je de naam van een kind wilt wijzigen, dan kun je dit doen door eerst een transformatie uit te voeren waarbij er combinaties van mannen en vrouwen een kind hebben. Dan ziet de tabel er als volgt uit:





Figuur 3 – Doelpopulatie onbewerkt, met uitvergroting.

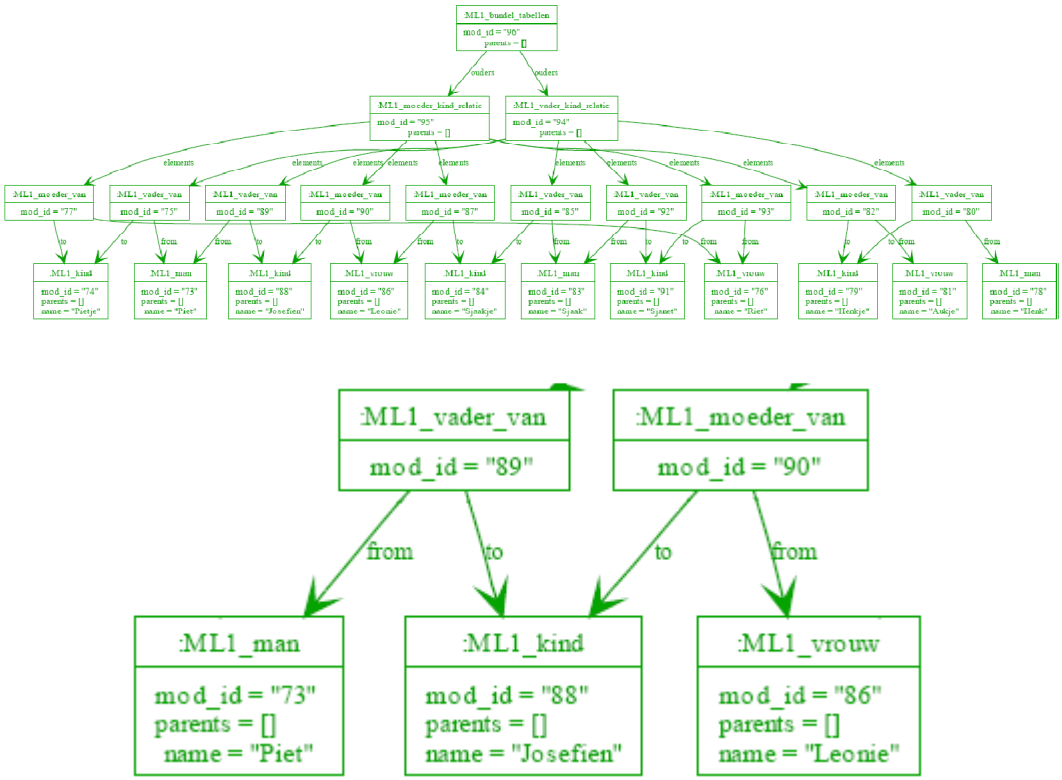
Nu hoef je de naam van het kind maar op één plek te veranderen. Een voorbeeld wijziging is toegepast in onderstaande figuur:



Figuur 4 – Doelpopulatie bewerkt, met uitvergroting.

Merk op dat de figuur van de brontabel insinueert dat een kind maar op één plek staat. Wanneer je de tabellen ziet zul je opmerken dat hij wel degelijk op twee verschillende plekken staat.

In praktijk wil je natuurlijk alle wijzingen toepassen voordat je de transformatie weer ongedaan maakt. In dit beknopte voorbeeld zijn we echter klaar. Door de transformatie weer terug te draaien krijgen we de oorspronkelijke tabellen weer terug met de wijziging toegepast (zie figuur 5). In dit eenvoudige voorbeeld zal je al snel het idee hebben dat dit een hoop moeite is voor een kleine verandering. In de praktijk is het echter allemaal lang zo overzichtelijk niet en kunnen eenvoudig inconsistenties voordoen omdat dubbele informatie niet compleet aangepast wordt.



Figuur 5 – Bronpopulatie bewerkt, met uitvergroting.

Merk op dat in dit voorbeeld het veranderde gegeven als sleutel fungeert tussen de twee losse tabellen. Dit is geen beperking van deze methode. Het is namelijk mogelijk om alle informatie binnen een database op deze manier te wijzigen.

3 Waarom datatransformatie in MT?

MT is een modeltransformatietaal welke als Domain Specific Language (DSL) opereert binnen de programmeertaal Converge. Meer over Converge is te vinden in hoofdstuk 4 . DSL wil zeggen dat het een programma is wat problemen oplost binnen een specifiek domein. Het specifieke domein is in het geval van MT het transformeren van modelstructuren. MT is geschreven in de programmeertaal Converge. Converge is een moderne taal welke eigenschappen heeft van Python, Haskell, Icon en Smalltalk. Deze taal is ervoor geschikt om uitgebreid te worden met zelfgemaakte semantiek, hierdoor kon MT eenvoudig binnen Converge geïmplementeerd worden. Laurence Tratt is de hoofd-verantwoordelijke geweest bij zowel het maken van Converge als het maken van MT. In het kort wordt MT door de maker[5] beschreven als: "MT is een MTL welke laat zien hoe een QVT achtige taal uitgebreid kan worden met nieuwe pattern matching constructen, hoe informatietraces automatisch geconstrueerd en gevisualiseerd kunnen worden en hoe het getransformeerde model ontdaan wordt van irrelevante onderdelen" (Meer informatie over QVT is te vinden in [7]).

3.1. Voordelen

Hoewel MT academisch tot stand is gekomen en daardoor niet van alle gemakken voorzien is heeft het toch wat sterke punten. Een voordeel van MT is dat het heel vrij is met de toepassing van Converge code binnen de transformatiebeschrijvingen. Hierdoor geniet de gebruiker van meer mogelijkheden dan normaalgesproken mogelijk zijn met een transformatietaal. Een ander voordeel van MT is dat er buitengewoon veel tracing informatie beschikbaar is met pakkende visualisaties. Hierdoor is het makkelijk te overzien of de transformatie doet wat je verwacht. Deze tracing informatie wordt door MT zelf gegenereerd en hoeft dus nauwelijks aandacht van degene die de transformatie beschrijft.

3.2. Nadelen

De taal is ontworpen voor onderzoeksdoelstellingen wat met zich meebrengt dat er eigenlijk geen ontwikkelingen zijn om MT vriendelijker in het gebruik maken. De maker heeft niet direct de intentie om MT te herschrijven naar de nieuwste versie van Converge. Dit brengt als nadeel met zich mee dat de versie waarin MT wel ondersteund is (Converge 0.8) een minder prettige Converge compiler is. Zo is de nieuwste versie van Converge wel geschikt voor Windows en de oude niet. En bovendien is de installatie van Converge 0.8 omslachtig doordat er gebruik gemaakt wordt van andere (verouderde) software.

3.3. Alternatieve talen

Het doorgronden van andere talen valt buiten de scope van deze scriptie. Ik zal echter in het kader van achtergrond informatie toelichten wat potentiële alternatieven zijn.

3.3.1. QVT

QVT is een definitie, maar geen volledige implementatie. Zo is MT ook ontstaan met eigenschappen van QVT. Enkele bedrijven die gebruik maken van de QVT standaard zijn Borland, Compuware, France Telecom, Hewlett Packard en Sun Microsystems. Achtergrond informatie over QVT in het kader van MT is te vinden in [7]

3.3.2. ATL

ATL is geimplementeerd als plugin voor Eclipse. Het werkt dan ook vanzelfsprekend in combinatie met Java programmas. Daarnaast zijn er minstens 256 Metamodellen beschikbaar. Dit wil zeggen dat vele talen ondersteund worden. Denk hierbij aan talen als C(++), DotNET, PetriNet, HTML, JAVA, MSproject, MySQL, QVT, sysML, UML, XML etc. Deze taal zal dan ook op zijn minst in staat zijn platte transformaties uit te voeren waarbij slechts de taal veranderd uit te voeren. Meer over ATL zie [8]

3.3.3. VIATRA 2

VIATRA 2 profileert zich als zijnde General Purpose support bij het maken van model transformaties. Hierbij moet je denken aan model-analyse transformaties, model naar model transformaties, model naar code transformaties en een tool integration platform. Het werkt met behulp van Graph patterns. Een andere taal met dezelfde grafen theorie is GReAT. Meer over VIATRA2 zie [9]

3.3.4. Tefkat & Kermeta

Tefkat en Kermeta zijn afgeleiden van QVT welke net als ATL een plugin binnen Eclipse vormen. Ondanks dat ik ze in één adem noem zijn ze onafhankelijk van elkaar tot stand gekomen. Meer over Tefkat zie [10] meer over Kermeta zie [11] .

3.3.5. LX

LX is een open source general purpose programmeertaal welke uitermate geschikt is voor patroonherkenning. Hierdoor zijn transformaties eenvoudiger te definiëren dan in talen als C++ of Java. Verder bestaat er nog MOLA welke een grafische laag is bovenop LX. Voor meer over Mola zie [12] .

4 Converge

Converge [6] is een moderne programmeertaal welke eigenschappen heeft van Python, Haskell, Icon en Smalltalk en is gemaakt door Laurence Tratt. Dit zijn allemaal talen waarvan ik slechts de naam ken. Aangezien het een object-georienteerde taal is zal ik hem impliciet vergelijken met voor mij bekende talen als C++ en Java. Verschillen in notatie zijn een praktisch probleem voor mij, maar slechts een triviaal probleem. Daarom zal ik deze niet nader toelichten. De echte afwijkingen zal ik wel benoemen en evalueren wat de relevantie is van deze eigenschappen bij het transformeren van datastructuren.

4.1. Functie verwijzingen

Converge gaat met functies om net als met variabelen. Zo kun je een nieuwe naam toewijzen aan een bestaande functie net als dat je een variabele kunt toewijzen aan een andere variabele. Een interessante toepassing van deze eigenschap is dat je een functie mee kunt geven aan een andere functie. Hierdoor kun je een functie parameteriseerbaar maken op een hoger niveau. Deze eigenschap zich als zinvol bewijzen bij het algemeen houden van de transformatie, waarbij de specifieke kenmerken van de bron en doeltaal van de data niet in het transformatiealgoritme opgenomen hoeven te worden. Deze eigenschap kan dus goed bijdragen aan portabiliteit van het transformatiealgoritme.

4.2. Compile-time meta programming

De taal ondersteunt ook meta programming (zie H3 van Tratt [6]). Zo heb ik een voorbeeld gezien waarbij compile time berekeningen worden uitgevoerd zodat ze runtime gebruikt kunnen worden zonder steeds weer dezelfde berekening te hoeven doen. Het voorbeeld genereerde expliciete code voor een specifieke printf aanvraag. Namelijk een string gevolgd door een integer en weer gevolgd door een string. Hiervoor werd code gegenereerd welke altijd de eerste controleert op een string, de tweede op een integer en de derde weer op een string. De besparing zit hier in het niet steeds weer hoeven bepalen wat de geeiste typen van de parameters zijn. Blijkbaar is hier grote tijdswinst te behalen.

Verder is er een sprekend voorbeeld te vinden in bijlage A.1. Hierbij wordt de functie ‘power3’ gegenereerd door middel van de functieaanroep ‘mk_power(3)’. In het kort komt het erop neer dat je met een functie een andere functie kunt genereren. Er wordt dus in dit voorbeeld compile time een functie ‘power3’ gegenereerd met gebruik van de functie ‘mk_power’. Op run time wordt er gebruik gemaakt van power3 alsof het altijd al een bestaande functie geweest is.

MT op zichzelf is gerealiseerd met meta programming. Dit wil zeggen dat de transformatiecode die je geschreven hebt, met behulp van meta programming verwerkt wordt.

4.3. Datastructuren

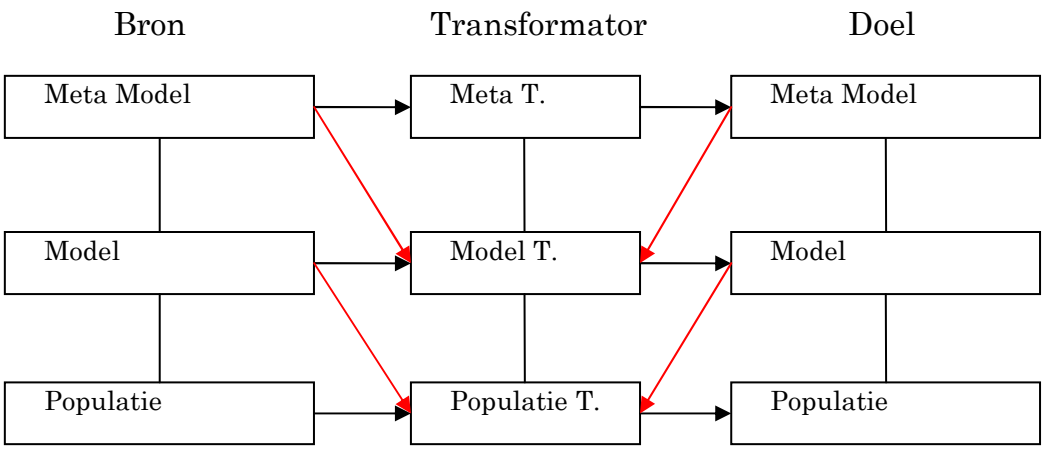
Net zoals MT een DSL binnen Converge is, is TM een DSL binnen converge. Ook TM wordt dus met behulp van compile-time meta programming gerealiseerd. TM staat voor Typed Modelling, dat wil zeggen dat je zelf vast kunt leggen hoe een datastructuur eruit ziet. TM ondersteunt UML achtige modelleertalen en hun instanties weergegeven te worden in een “flat namespace” (zie Tratt [6]). Voor deze scriptie is slechts van belang dat MT TM gebruikt om de bron en doelmodellen vast te leggen.

5 Transformatie

We willen transformaties uitvoeren welke de structuur van de data aanpassen, maar dat er geen informatie verloren gaat of bij verzonnen wordt. Het is dus zaak om eerst een model te maken van de informatie die we hebben. Normaalgesproken wordt dit model bepaald aan de hand van de te transformeren informatiebron(nen), maar in dit geval kunnen twee verschillende structuren afgeleid worden van een te kiezen model.

Voor een transformatie moet je eerst bepalen wat het abstractieniveau van de transformatie moet zijn. Zo kun je populatie, model en meta model transformaties uitvoeren. De interessantste transformatie is die van de populatie. Dit komt uit het feit dat populaties vaker dan modellen gewijzigd worden. Modellen op zijn beurt worden weer vaker gewijzigd dan meta modellen.

Hieronder staat een overzicht van de verschillende abstractielagen. De transformatie die ik ga behandelen in hoofdstuk 5 en 6 is een populatie transformatie. Hierbij heb je een bronmodel, doelmodel, populatie transformator en een beginpopulatie nodig. Wanneer je deze hebt vastgesteld kun je de transformatie uitvoeren waarna je de doelpopulatie verkrijgt.

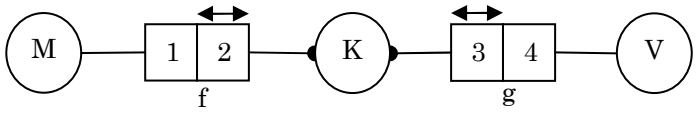


Figuur 6 – Overzicht abstractielagen bij transformaties

5.1. Conceptuele model

Het conceptuele model is een begrip wat los staat van het schema in figuur 6. Het idee van het conceptuele model is dat er een heel domein is aan representaties (tabellen) welke dezelfde informatie bevatten. Dit wil zeggen dat zolang je binnen het domein transformeert van een representatie naar een andere dat er geen informatie kwijtraakt, maar ook geen informatie bij komt. De reden dat ik het conceptuele model introduceer op dit punt is dat ik alleen transformaties behandel die binnen hetzelfde domein representaties opleveren.

Het conceptuele model van de te beschouwen transformatie is een betrekkelijk eenvoudig model, later volgen complexere modellen en transformaties.



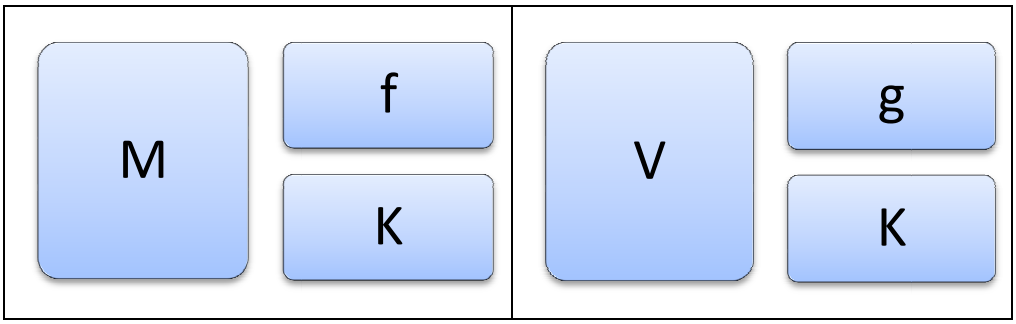
M: Man	1: is vader van <2>
K: Kind	2: is kind van <1>
V: Vrouw	3: is kind van <4>
	4: is moeder van <3>

Figuur 7 – Conceptuele Model.

5.2. Brontabel modellen

Als beginpopulatie heb je twee tabellen. Namelijk een tabel met alle kinderen bij de juiste vader geplaatst en eenzelfde tabel voor de moeders. In deze paragraaf wordt beschreven hoe het model van de invoer eruit ziet, de populatie zal zich aan dit model moeten conformeren. Dit populeren zal geschieden in het volgende hoofdstuk. Merk op dat in dit voorbeeld de beginsituatie identiek is aan het conceptuele model. Dit hoeft niet zo te zijn.

In het onderstaande model heb je twee afzonderlijke modellen welke bestaan uit een aantal symbolen. In het eerste model heb je een man (M) welke via een relatie f aan een kind (K) vast zit. Hiermee wordt niets anders bedoeld dan dat je koppels hebt van mannen met kinderen. Voor de vrouw (V) geldt hetzelfde verhaal voor het rechter model. Hierbij geeft g de relatie aan. Merk bij beide modellen op dat er een uniqueness constraint zit op K.

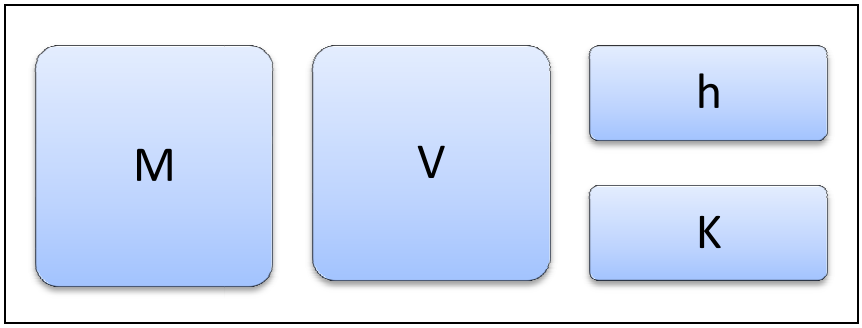


Figuur 8 – Modellen Brontabellen.

5.3. Doeltabel model

De gewenste eindtabel is te weten welke mannen en vrouwen samen kinderen hebben. Deze informatie is beschikbaar in het bronmodel, maar op een zodanige manier dat het niet direct op te vragen is. Wanneer er behoefte is aan koppels is dit een gerechtvaardigde transformatie. Merk ook hierbij op dat dit het model is waaraan de doelpopulatie zich dient te conformeren.

Het onderstaande model bestaat uit mannen(M), vrouwen(V) en kinderen (K). Hierbij wordt er per man/vrouw combinatie een kind toegewezen. Dit wordt door de relatie h aangegeven. Merk op dat de uniqueness constraint over K nodig is om de transformatie van het bronmodel naar het doelmodel te kunnen doen. Anders is het niet te bepalen welke man met welke vrouw een kind heeft (als er meerdere kinderen zijn met dezelfde naam).



Figuur 9 – Model Doeltabel.

5.4. Conceptuele model tabellen

Zoals al eerder vermeld zijn de brontabellen in dit voorbeeld gelijk aan het conceptuele model. Wanneer dit niet het geval is kan het wel wijs zijn om de brontabellen eerst te converteren naar het conceptuele model. Dit houdt in dat je tabellen maakt voor ieder feittype in het conceptuele model. Het voordeel van deze methode is dat je als er een nieuwe representatie bijkomt dat er alleen een conversie van/naar het conceptuele model gemaakt hoeft te worden. Hierdoor heb je geen last van toenemende hoeveelheid vertalingen. Als je n talen hebt moet je normaliter *(n+1)*n/2* vertalers hebben in het geval dat je alles direct naar elkaar toe wilt vertalen. Als je een tussentaal neemt (in dit geval het conceptuele model), is dit aantal te verminderen naar *n*. Namelijk voor iedere taal een vertaler naar het conceptuele model. Merk op dat ik bij het bepalen van van het aantal vertalers een vertaler heb gedefinieerd als een vertaler die zowel a -> b doet als b -> a.

6 MT transformatie code

Nu we bepaald hebben wat voor modeltransformatie we willen doen is het zaak deze te beschrijven in MT. Zo moeten de brontabellen en de doeltabel beschreven worden en daarna de transformatieregels.

6.1. Het Bronmodel

Het bronmodel is het model van de brontabellen (zie 5.2). In dit specifieke voorbeeld beschrijven we het model wat hoort bij de tabellen vader en moeder (populatie). Hiervoor worden de klassen mannen,vrouwen en kinderen gebruikt om kolomnamen te definieren. De vader tabel bestaat uit mannen met kinderen en de moeder tabel bestaat uit vrouwen met kinderen. Voor beide tabellen geldt dat kinderen uniek zijn binnen de tabel. In de bijlage (A.2) is te vinden hoe de MT code van het bronmodel eruit ziet. Ik zal wat onderdelen ervan beschouwen om wat meer helderheid te verschaffen wat de code betekent. Dit doe ik aan de hand van codefragmenten.

Om aan te geven dat een kolom bestaat, moet er een kolom aangemaakt worden. Dit is als volgt gedefinieerd:

```
class ML1_vrouw extends ML1_ouder_kind_relatie {
                parents : Seq(ML1_vrouw);
        }
```

Hierbij is de vrouw gedefinieerd welke een extentie is van de ouder_kind_relatie. Dit wil zeggen dat ze de eigenschappen heeft van ouder_kind_relatie. Deze klasse bestaat uit slechts een string welke een naam dient te bevatten. Nu moet een vrouw dus ook een naam bevatten.

Naast het vaststellen van kolomnamen wil je vastzetten dat verschillende kolommen verband met elkaar houden. Dit komt in onderstaande codefragment aan bod.

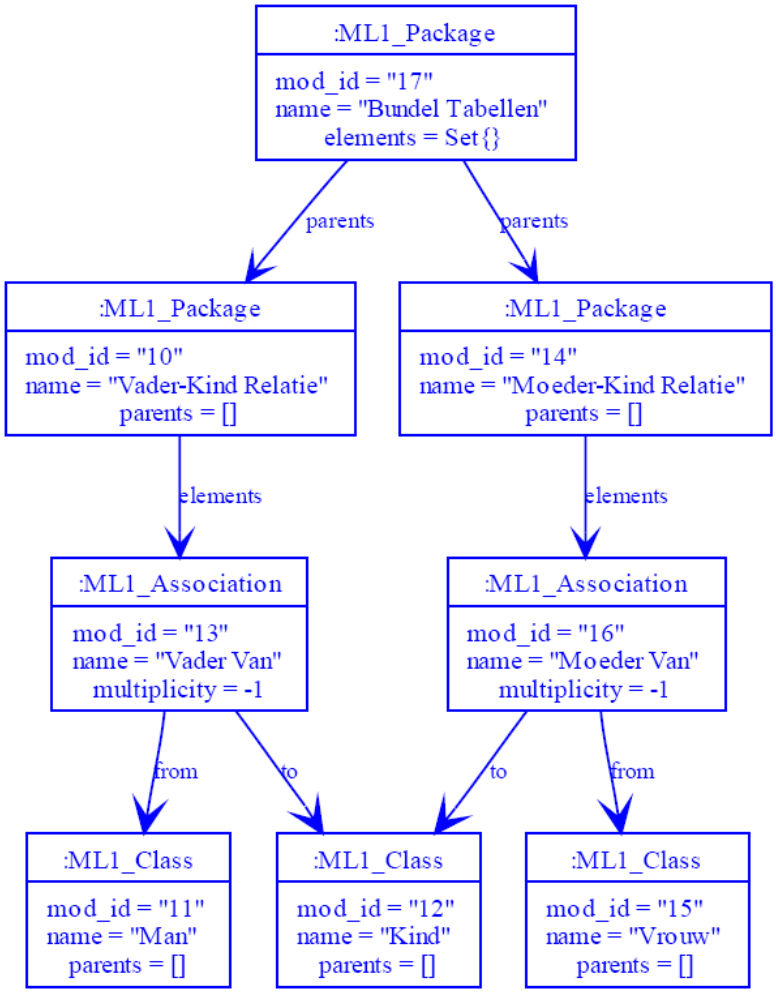
```
class ML1_moeder_van {
                from : ML1_vrouw;
                to : ML1_kind;
        }
```

Hier is de relatie gedefinieerd tussen een vrouw en een kind, merk op dat het kind op dezelfde manier gedefinieerd is als de vrouw. Een moeder_van kun je zien als een rij in een tabel. Met dit vastgesteld moet er nog voor gezorgd worden dat er meerdere rijen in de tabel mogen staan. Daarvoor zorgt onderstaande codefragment.

```
class ML1_moeder_kind_relatie {
                elements : Set(ML1_ouder_kind_relatie);
                parents : Seq(ML1_moeder_kind_relatie);

                inv unique_names:
```

```
                elements->forAll(a1 a2 |
                        a1 != a2 implies a1.name != a2.name)

        operation allElements():
                elements + parents->collect(p e = Set{} |
                        e + p.allElements())

}
```

Hierbij is elements de verzameling van moeder_van entries (rijen in de tabel). Daarnaast staat in deze class een check op dubbele namen en een operatie om alle elementen boven water te krijgen. Merk op dat er een "bundel tabellen" object voorkomt in de definitie. Deze is noodzakelijk om twee tabellen te gebruiken als input voor een transformatie. Ter verduidelijking hieronder een visualisatie van het bronmodel.



Figuur 10 – Metamodel Bronmodel.

6.2. Het doelmodel

Het doelmodel is het model van de doeltabellen (zie 5.3). In dit specifieke voorbeeld beschrijven we het model wat hoort bij de tabellen ouders. Deze tabel bestaat uit mannen en vrouwen met kinderen (ieder kind is uniek in de tabel). In de bijlage (A.3) is te vinden hoe de MT code van het doelmodel eruit ziet. Deze is opgebouwd uit dezelfde bouwstenen als het bronmodel en zal daarom niet verder toegelicht worden.

6.3. Transformatie

Het gaat om een populatie transformatie, de modellen worden beschreven om de transformatieregels compleet te maken. De transformatie moet van een bundeling van twee oudertabellen (vaders en moeders) een samengevoegde tabel maken waarbij de kinderen uniek zijn en daarmee de link tussen vaders en moeders bepaald wordt. In de bijlage (A.4) is te vinden hoe de MT code van de transformatie eruit ziet. Voor de volledigheid heb ik ook toegevoegd hoe de transformatie weer omgekeerd kan worden (bijlage A.5).

Een belangrijk punt van deze transformatie is het volgende codefragment

```
rule Bundel_To_Ouderkindrelatie:

        srcp:
                (ML1_bundel_tabellen, <ml1_bundel>)[ouders == <o>, parents == <p>]

        tgtp:
                (ML2_ouders_kind_relatie)[elements := e, parents := p]

        tgt_where:
                ouderlijst := []
                for ouder := o.iterate():
                        ouderlijst.append([ouder])

                vaders := self.transform(ouderlijst[0])
                moeders := self.transform(ouderlijst[1])

                e := Set{}

                for vader := vaders.iterate():
                        e.add(vader)

                for moeder := moeders.iterate():
                        for ouder := e.iterate():
                                if (ouder.kind == moeder.kind):
                                        ouder.moeder := moeder.moeder
```

Het opmerkelijke aan dit codefragment is dat het niet triviale omzettingen betreft. Het transformeert eerst de vader en de moedertabel waarna hij ze samenvoegt. Het werkt als volgt. Er wordt een oudertabel aangemaakt waarbij de vaders met hun kinderen ingevuld staan met dummy waarden voor de moeder. Daarna wordt door de

oudertabel heengelopen van de moeders, daarbij wordt de dummy waarde in de oudertabel vervangen door de echte naam van de moeder. Hierbij geldt het kind als unieke sleutel.

6.4. De executie

Nu we alles gedefinieerd hebben in MT code, staat niets ons in de weg om de transformatie daadwerkelijk te executeren. Ik heb een populatie aangemaakt voor de begintabellen (vadertabel en moedertabel). Deze is volledig te vinden in bijlage A.7.

Het maken van de invoer ziet er overzichtelijk uit, maar toch geef ik voor de volledigheid een korte beschrijving van de verschillende commando's die je ziet.

```
man1 := ML1.ML1_man("Piet")
```

Hierbij wordt een nieuwe man aangemaakt genaamd "Piet".

```
vaders := ML1.ML1_vader_kind_relatie()
vaders.elements.add(ML1.ML1_vader_van(man1, kind1))
```

Daarna wordt een entry aangemaakt die een relatie tussen een vader en zijn kind aangeeft. Daaraan wordt man1 en kind1 toegevoegd.

```
oudertabellen := ML1.ML1_bundel_tabellen()
oudertabellen.ouders.add(vaders)
```

Vervolgens zien we dat de oudertabellen gedefinieerd worden waar de vaders aan toegevoegd worden.

Als je de code uit bijlage A.7 bekijkt zul je zien dat hierna de oudertabellen de tabellen vaders en moeders bevat. Welke op hun beurt weer entries bevatten en deze weer bestaan uit de ouder met een kind. Hoe de invoer er precies uitziet is te zien in bijlage A.6 onder input, daarnaast is een visuele versie te zien in bijlage B.1.

Deze populatie willen we transformeren naar een tabel waarbij de ouders gezamenlijk staan met hun gemeenschappelijke kind. De transformatie trace is te zien in bijlage B.3. Na de executie van de transformatie krijg je een tabel zoals te zien in bijlage A.6 onder output, daarnaast is er een visuele versie te zien in bijlage B.2. Dit is de daadwerkelijke generatie van MT.

6.4.1. De transformatie terug

Verder heb ik om de transformatie compleet te maken een transformatie de andere kant op gemaakt en uitgevoerd. Deze gebruikt de doeltabel om de brontabellen te genereren, zie bijlage A.6 onder Backwards Output. Hierbij is er ook weer een opmerkelijk stukje code.

```
rule ouders_To_bundeltabellen:

        srcp:
                (ML2_ouders_kind_relatie, <ml2_okr>)[elements == <e>, parents == <p>]

        tgtp:
                (ML1_bundel_tabellen)[ouders := Set{vaders, moeders}, parents := p]

        tgt_where:

                tempvaders := Set{}
                tempmoeders := Set{}

                for ot := e.iterate():
                        tempvaders.add((ML1_vader_van)[from :=
                                self.transform([ot.vader]), to := self.transform([ot.kind])])
                        tempmoeders.add((ML1_moeder_van)[from :=
                                self.transform([ot.moeder]), to := self.transform([ot.kind])])

                vaders := (ML1_vader_kind_relatie)[elements := tempvaders]
                moeders := (ML1_moeder_kind_relatie)[elements := tempmoeders]
```

Hierbij worden de vader en moedertabellen tegelijkertijd gevuld met informatie uit de oudertabellen. Wanneer deze volledig omgezet zijn worden ze ineens in de gewenste structuur gegoten. De volledige code is te zien in bijlage A.5.

De traceinformatie van deze transformatie terug is te vinden in B.4.

7 Problemen tegengekomen

Tijdens het werkend krijgen van MT ben ik tegen een aantal punten aangelopen. Ik zal deze in dit hoofdstuk toelichten.

7.1. Waar is MT?

Mijn eerste ontmoeting met MT was via rapporten. Hierbij bleek al snel dat MT een onderdeel was van de programmeertaal Converge. Converge bleek een eigen website te hebben waar de compiler voor verschillende machines waaronder windows beschikbaar was. Na het aan de praat krijgen van de nieuwste versie (1.0) van Converge en het uitvoeren van wat eenvoudige converge voorbeelden was het tijd om een MT programma uit te proberen. Aangezien er bij de compiler geen voorbeelden zaten voor MT besloot ik de voorbeelden uit verschillende rapporten te proberen. Dit kreeg ik niet aan de praat.

Aangezien converge gemaakt was door een universitair onderzoeker leek het me geschikt de maker eens te vragen per email wat het probleem kon wezen. Deze wist mij te vertellen dat MT nog niet geport was naar Converge 1.0. Dat was dus de verklaring waarom mijn MT voorbeelden niet werkten. Converge 0.8 is de laatste versie die MT ondersteunt, deze informatie was niet online te vinden overigens.

7.2. Verouderde Converge

Wegens voorgaande probleem besloot ik even Converge 0.8 te installeren. Er bleek een aanzienlijk verschil te bestaan tussen beide versies. Het eerste belangrijke verschil was dat er geen echte windows versie bestond. Dit terwijl ik op mijn werkplek alleen een windows machine heb staan. Er bestond wel de mogelijkheid om via Cygwin converge te gebruiken. Dit was de meest logische oplossing. Voor het compilen van Converge programma's en de Converge compiler bleek het programma bmake nodig te zijn. Voorheen stond er een bmake op de site van Converge, maar deze bleek weggehaald te zijn. Dit omdat Converge 0.8 een verouderde versie was. Aangezien ik weinig van bmake afwist en bovendien weinig kon vinden wat verband hield met bmake en Cygwin besloot ik de maker te vragen om advies. Deze wist mij uiteindelijk de sourcecode van bmake ter beschikking te stellen. Deze had ik vrij snel aan de praat en daarmee ook Converge.

7.3. Visualisering trace informatie

Zoals aangegeven werkte converge inmiddels. Bij deze versie zijn ook voorbeelden van MT opgenomen wat natuurlijk mooi is om eens uit te proberen. De voorbeeld-transformaties werden correct uitgevoerd, maar na het uitvoeren ervan werd er

gepoogd een executable *dot* en een executable *gv* aan te roepen met wat parameters erbij. Ik kende GhostView al en besloot daardoor dat het aannemelijk was dat dot een bestand aanmaakt wat geopend wordt door GhostView. Maar het programma dot werd niet herkend en GV opende een leeg bestand. Na wat gezocht te hebben op internet kwam ik tot de conclusie dat dot een onderdeel is van het pakket GraphViz. Dit is een pakket wat informatiestructuren kan omzetten naar grafische weergaven. Voor de zekerheid heb ik dit nog nagevraagd aan de maker van Converge, deze kon dit beamen. Na vele pogingen op van allerlei manieren dot aan de praat te krijgen moest ik toegeven dat het programma niet geschikt was voor Cygwin.

7.4. Wat zijn de regels voor metamodellen?

Er zijn verschillende papers geschreven over hoe je transformaties kunt definieren in MT. Helaas kom je tot de constatering dat de beschrijving van de metamodellen niet beschikbaar is en je op jezelf aangewezen bent deze te doorgronden. Hierbij kunnen de beschikbare voorbeelden je wat op weg helpen. Verder komt het gebrek aan toelichting ook weer terug bij het maken van de transformatieregels. Hoewel deze beschreven worden in een aantal papers, blijkt er meer informatie nodig voor het opstellen van een eigen transformatie. Dit heeft wellicht temaken met het feit dat je vrij bent converge code te gebruiken binnen de transformatiebeschrijving. Hierdoor is het bij complexere transformaties nodig om ook ruime Converge kennis in huis te hebben.

7.5. Hoe werkt converge?

Zoals net verteld is converge kennis ook vereist bij het opstellen van niet triviale transformaties. Het probleem is echter dat mijn doel niet was converge programmas te maken, maar het opstellen van transformaties in MT. Het maken van converge programmas an sich is namelijk een uitdaging. Want net als MT heeft Converge een gebrek aan een complete manual. Het leren van alle concepten van Converge zou mij tever van mijn doel af doen gaan.

7.6. Wederom GraphViz

Toen ik in staat was zelf een transformatie te maken van een populatie naar de andere kwam het gevoel weer terug dat het mogelijk moest zijn zowel de modellen grafisch weer te geven als de transformatie. Dit zou in zodanige wijze mijn scriptie kunnen verduidelijken dat ik nog een poging wilde wagen. Dit keer pakte ik het rigoreus aan. Namelijk het proberen op een andere architectuur. Eerst heb ik gepoogd het pakket aan de praat te krijgen onder Solaris, dit bleek geen succes te zijn. Zowel Converge als GraphViz bleken niet te werken onder Solaris. Hierna werd mij toegang geboden tot een Linux server. Hiermee hoopte ik dat het zonder

problemen zou functioneren. Het bleek ook hier echter niet mogelijk om converge aan de praat te krijgen. GraphViz bleek echter wel goed te functioneren op deze architectuur. Toen kwam ik op het idee om eens te kijken of ik de bestanden kon bemachtigen die gegenereerd worden door MT (in Cygwin) om de visualisatie te verwezelijken. Deze bleken gemaakt te worden in een tijdelijke map en ook niet verwijderd te worden. Na wat aanpassen van parameters bleek het mogelijk te zijn om de tijdelijke bestanden in te voeren in dot (op Linux server) en daarvan pdf bestanden te maken. Het resultaat is te bewonderen in bijlage B.

7.7. Abstraherend Transformeren

Wanneer je een modeltransformatie definieert in MT heb je uiteindelijk een visuele versie van je bronmodel, van je doelmodel en een visualisatie van de transformatie tussen beide. Als je dit hebt ben je in praktijk nog lang niet klaar. Het transformeren van modellen opzich is meestal een middel en geen doel opzich. Je bent geïnteresseerd in het transformeren van populaties, want daar gaat veel tijd in zitten wanneer je dit handmatig moet doen. Nu wordt er in de paper van MT [5] geen woord gerept over populatie transformaties. Na bij Tratt geinformeerd te hebben kom ik tot de conclusie dat dit niet mogelijk is om automatisch modellen in MT te gebruiken bij transformaties op verschillende lagen van abstractie. Bij modeltransformatie is het namelijk de input en output, terwijl het bij populatietransformatie het als model van de invoer en model van de uitvoer is. Deze zijn op een andere manier gespecificeerd waardoor je handmatig de vertaalslag dient te maken.

8 Conclusies

Zoals aangegeven in de introductie (Hoofdstuk 1) heb ik de relevantie van het transformeren van één datastructuur in de andere beschouwd. In paragraaf 8.1 ga ik daar verder op in. De grootste uitdaging van deze scriptie bleek het werkend te krijgen van MT en daarin een geschikte transformatie te specificeren en uit te voeren. In paragraaf 8.2 ga ik hier verder op in.

8.1. Relevantie modeltransformaties

Modeltransformaties zijn uitermate zinvol wanneer er een andere representatie gewenst is dan beschikbaar is. Denk hierbij aan de tijd/ruimte afwegingen en aan de veelgebruikte zoekoperaties (zie 2.1). Verder kun je denken aan transformaties naar verschillende representaties voor verschillende doelgroepen (zie 2.2). Ook kunnen ze bij ingewikkelde databases met veel redundantie het updaten vereenvoudigen (zie 2.3).

8.2. Relevantie MT

MT heeft laten zien dat hij bruikbaar is om modeltransformaties uit te voeren. Het is mij namelijk gelukt om een populatie transformatie uit te voeren in MT (zie hoofdstuk 6 Hierbij is zijn sterke punt dat het goed te zien is in de visualisatie welke transformatieregels hij op welke instanties toegepast heeft. Bij het maken van deze scriptie zijn de visueel ondersteunende middelen zowieso erg prettig om de voorbeelden wat levendiger te maken (zie bijvoorbeeld 2.3.1).

Ik heb als nadeel ervaren dat het niet mogelijk is om een modeltransformatie uit te voeren en deze modellen direct te gebruiken als (meta-)modellen voor populatie transformaties. Het gevolg is dat je een modeltransformatie kunt specificeren, waarna je mooi gevisualiseerd een overzicht hebt hoe precies het ene model in de andere wordt omgeschreven. Maar wanneer je vervolgens populaties van de betreffende modellen wilt transformeren dit niet automatisch mogelijk is. Je zult dan handmatig de modellen moeten specificeren voordat je een populatie kunt transformeren. Kortweg mist de transformatiestap tussen de verschillende lagen van abstractie (zie 7.7).

9 Referenties

[1] S. Cluet, C. Delobel, J. Siméon, K. Smaga. Your mediators need data conversion! *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, Seattle, Washington, United States, Pages: 177 – 188, 1998.

[2] N. C. Shu, B. C. Housel, V. Y. Lum. CONVERT: a high level translation definition language for data conversion. *Communications of the ACM*, Volume 18, Issue 10, Pages 557 – 567, October 1975.

[3] V. Y. Lum, N. C. Shu, B. C. Housel. A General Methodology for Data Conversion and Restructuring. IBM Journal of Research and Development, Volume 20, Number 5, Page 483, 1976.

[4] P. van Bommel. Data models and transformations. Lecture Notes, Radboud University Nijmegen, February 2008.

[5] L. Tratt. Model transformations in MT. *Science of Computer Programming*, Volume 68, Issue 3, Pages 196-213, October 2007.

[6] L. Tratt. The Converge programming language. *Technical Report TR-05-01*, Department of Computer Science, King's College London, February 2005.

[7] QVT-Partners. First revised submission to QVT RFP, August 2003. OMG document ad/03-08-08.

[8] J. Bézivin , E. Breton, G. Dupé, P. Valduriez. The ATL Transformation-based Model Management Framework. *RESEARCH REPORT, N° 03.08*, IRIN, Université de Nantes, September 2003.

[9] A. Balogh, D. Varró. Advanced model transformation language constructs in the VIATRA2 framework, *Proceedings of the 2006 ACM symposium on Applied computing, Model transformation (MT 2006)*, Pages 1280 – 1287, 2006.

[10] M. Lawley, J. Steel. Practical Declarative Model Transformation with Tefkat, *Satellite Events at the MoDELS 2005 Conference*, Volume 3844/2006, pages 139 – 150, January 2006.

[11] J-R Falleri, M. Huchard, C. Nebut. Towards a traceability framework for model transformations in Kermeta, ECMDA-TW Workshop, 2006.

[12] A Kalnins, J Barzdins, E Celms. Model Transformation Language MOLA, *Model Driven Architecture*, Volume 3599/2005, pages 62 – 76, August 2005.

A. Bijlage SourceCode

A.1. Dynamisch functie definieren.

```
func expand_power(n, x):

    if n == 0:
        return [| 1 |]
    else:
        return [| $c{x} * $c{expand_power(n - 1, x)} |]



func mk_power(n):

    it := [|
        func (&x):
            return $c{expand_power(n, [| &x |])}
    |]

    Sys::println("mk_power(", n, ") generated:\n")
    Sys::println(CEI::pp_itree(it))

    return it



power3 := $<mk_power(3)>



func main():
            Sys::println("3^3 = ", power3(3))
```

A.2. Het bronmodel

```
$<TM.model_class>:

        abstract class ML1_ouder_kind_relatie {}

        class ML1_vader_kind_relatie extends ML1_ouder_kind_relatie {
                elements : Set(ML1_ouder_kind_relatie);
                parents : Seq(ML1_vader_kind_relatie);

                operation allElements():
                        elements + parents->collect(p e = Set{} | e + p.allElements())
        }

        class ML1_moeder_kind_relatie extends ML1_ouder_kind_relatie {
                elements : Set(ML1_ouder_kind_relatie);
                parents : Seq(ML1_moeder_kind_relatie);

                operation allElements():
                        elements + parents->collect(p e = Set{} | e + p.allElements())
        }

        class ML1_bundel_tabellen extends ML1_ouder_kind_relatie {
                ouders : Set(ML1_ouder_kind_relatie);
                parents : Seq(ML1_ouder_kind_relatie);
        }

        class ML1_man extends ML1_ouder_kind_relatie {
                name : String;
                parents : Seq(ML1_man);
                inv nonempty_name:
                        name != null and name.len() > 0
        }

        class ML1_vrouw extends ML1_ouder_kind_relatie {
                name : String;
                parents : Seq(ML1_vrouw);
                inv nonempty_name:
                        name != null and name.len() > 0
        }

        class ML1_kind extends ML1_ouder_kind_relatie {
                name : String;
                parents : Seq(ML1_kind);
                inv nonempty_name:
                        name != null and name.len() > 0
        }

        class ML1_vader_van extends ML1_ouder_kind_relatie {
                from : ML1_man;
                to : ML1_kind;
        }

        class ML1_moeder_van extends ML1_ouder_kind_relatie {
                from : ML1_vrouw;
                to : ML1_kind;
        }
```

A.3. Het doelmodel

```
$<TM.model_class>:

        abstract class ML2_general {
                name : String;

                inv nonempty_name:
                        name != null and name.len() > 0
        }

        class ML2_ouders_kind_relatie {
                elements : Set(ML2_ouders_van);
                parents : Seq(ML2_ouders_kind_relatie);

                operation allElements():
                        elements + parents->collect(p e = Set{} |
                                e + p.allElements())

        }

        class ML2_man extends ML2_general {
                parents : Seq(ML2_man);
        }

        class ML2_vrouw extends ML2_general {
                parents : Seq(ML2_vrouw);
        }

        class ML2_kind extends ML2_general {
                parents : Seq(ML2_kind);
        }

        class ML2_ouders_van {
                vader : ML2_man;
                moeder : ML2_vrouw;
                kind : ML2_kind;
        }
```

A.4. De transformatie

```
$<MT.mt>:
        transformation ML1_to_ML2

        rule Bundel_To_Ouderkindrelatie:

                srcp:
                        (ML1_bundel_tabellen, <ml1_bundel>)[ouders == <o>, parents == <p>]

                tgtp:
                        (ML2_ouders_kind_relatie)[elements := e, parents := p]

                tgt_where:
                        ouderlijst := []
                        for ouder := o.iterate():
                                ouderlijst.append([ouder])

                        vaders := self.transform(ouderlijst[0])
                        moeders := self.transform(ouderlijst[1])

                        e := Set{}

                        for vader := vaders.iterate():
                                e.add(vader)

                        for moeder := moeders.iterate():
                                for ouder := e.iterate():
                                        if (ouder.kind == moeder.kind):
                                                ouder.moeder := moeder.moeder

        rule vaders_To_ouders:

                srcp:
                        (ML1_vader_kind_relatie, <ml1_vkr>)[elements == <e>, parents == <p>]

                tgtp:
                        self.transform_all(e)

        rule moeders_To_ouders:

                srcp:
                        (ML1_moeder_kind_relatie, <ml1_mkr>)[elements == <e>, parents == <p>]

                tgtp:
                        self.transform_all(e)

        rule vadervan_To_oudersvan:

                srcp:
                        (ML1_vader_van, <ml1_vv>)[from == <f>, to == <t>]

                tgtp:
                        (ML2_ouders_van)[vader := self.transform([f]), kind := self.transform([t]),
                                        moeder := tv]

                tgt_where:
                        tv := (ML2_vrouw)[name := "V_Vrouw"]
```

```
rule moedervan_To_oudersvan:

        srcp:
                (ML1_moeder_van, <ml1_mv>)[from == <f>, to == <t>]

        tgtp:
                (ML2_ouders_van)[moeder := self.transform([f]), kind := self.transform([t]),
                                vader := tm]

        tgt_where:
                tm := ML2.ML2_man("V_Man")

rule man_To_man:

        srcp:
                (ML1_man, <ml1_m>)[name == <n>]

        src_when:
                n.len() > 0

        tgtp:
                (ML2_man)[name := n]

rule vrouw_To_vrouw:

        srcp:
                (ML1_vrouw, <ml1_v>)[name == <n>]

        src_when:
                n.len() > 0

        tgtp:
                (ML2_vrouw)[name := n]

rule kind_To_kind:

        srcp:
                (ML1_kind, <ml1_k>)[name == <n>]

        src_when:
                n.len() > 0

        tgtp:
                (ML2_kind)[name := n]
```

A.5. De transformatie terug

```
$<MT.mt>:
        transformation ML2_to_ML1

        rule ouders_To_bundeltabellen:

                srcp:
                        (ML2_ouders_kind_relatie, <ml2_okr>)[elements == <e>, parents == <p>]

                tgtp:
                        (ML1_bundel_tabellen)[ouders := Set{vaders, moeders}, parents := p]

                tgt_where:
                        tempvaders := Set{}
                        tempmoeders := Set{}

                        for ot := e.iterate():
                                tempvaders.add((ML1_vader_van)[from :=
                                        self.transform([ot.vader]), to := self.transform([ot.kind])])
                                tempmoeders.add((ML1_moeder_van)[from :=
                                        self.transform([ot.moeder]), to := self.transform([ot.kind])])

                        vaders := (ML1_vader_kind_relatie)[elements := tempvaders]
                        moeders := (ML1_moeder_kind_relatie)[elements := tempmoeders]

        rule man_To_man:

                srcp:
                        (ML2_man, <ml2_m>)[name == <n>]

                src_when:
                        n.len() > 0

                tgtp:
                        (ML1_man)[name := n]

        rule vrouw_To_vrouw:

                srcp:
                        (ML2_vrouw, <ml2_v>)[name == <n>]

                src_when:
                        n.len() > 0

                tgtp:
                        (ML1_vrouw)[name := n]

        rule kind_To_kind:

                srcp:
                        (ML2_kind, <ml2_k>)[name == <n>]

                src_when:
                        n.len() > 0

                tgtp:
                        (ML1_kind)[name := n]
```

A.6. Output MT

```
Input:
<ML1_bundel_tabellen@17109408
 ouders : Set{
  <ML1_vader_kind_relatie@19578640
   elements : Set{
    <ML1_vader_van@24093144
     from : <ML1_man@18105952
      name : "Piet"
      parents : []>
     to : <ML1_kind@19236752
      name : "Pietje"
      parents : []>>
    <ML1_vader_van@17559904
     from : <ML1_man@19687608
      name : "Henk"
      parents : []>
     to : <ML1_kind@19560320
      name : "Henkje"
      parents : []>>
    <ML1_vader_van@19737952
     from : <ML1_man@31371552
      name : "Sjaak"
      parents : []>
     to : <ML1_kind@19431088
      name : "Sjaakje"
      parents : []>>
    <ML1_vader_van@21450192
     from : <ML1_man@18105952
      name : "Piet"
      parents : []>
     to : <ML1_kind@28188192
      name : "Petra"
      parents : []>>
    <ML1_vader_van@18018504
     from : <ML1_man@31371552
      name : "Sjaak"
      parents : []>
     to : <ML1_kind@19495904
      name : "Sjanet"
      parents : []>>}
   parents : []>
  <ML1_moeder_kind_relatie@31546392
   elements : Set{
    <ML1_moeder_van@20395424
     from : <ML1_vrouw@38824808
      name : "Riet"
      parents : []>
     to : <ML1_kind@19236752
      name : "Pietje"
      parents : []>>
    <ML1_moeder_van@17717392
     from : <ML1_vrouw@22264176
      name : "Aukje"
      parents : []>
     to : <ML1_kind@19560320
      name : "Henkje"
```

```
          parents : []>>
        <ML1_moeder_van@17284152
          from : <ML1_vrouw@19813152
            name : "Leonie"
            parents : []>
          to : <ML1_kind@19431088
            name : "Sjaakje"
            parents : []>>
        <ML1_moeder_van@36136696
          from : <ML1_vrouw@19813152
            name : "Leonie"
            parents : []>
          to : <ML1_kind@28188192
            name : "Petra"
            parents : []>>
        <ML1_moeder_van@27514072
          from : <ML1_vrouw@38824808
            name : "Riet"
            parents : []>
          to : <ML1_kind@19495904
            name : "Sjanet"
            parents : []>>}
      parents : []>}
  parents : []>

Output:
<ML2_ouders_kind_relatie@21956440
  elements : Set{
    <ML2_ouders_van@18686344
      vader : <ML2_man@20112544
        name : "Piet"
        parents : []>
      moeder : <ML2_vrouw@18656248
        name : "Riet"
        parents : []>
      kind : <ML2_kind@33998592
        name : "Pietje"
        parents : []>>
    <ML2_ouders_van@23807864
      vader : <ML2_man@20159040
        name : "Henk"
        parents : []>
      moeder : <ML2_vrouw@21389896
        name : "Aukje"
        parents : []>
      kind : <ML2_kind@20378848
        name : "Henkje"
        parents : []>>
    <ML2_ouders_van@24592088
      vader : <ML2_man@22436064
        name : "Sjaak"
        parents : []>
      moeder : <ML2_vrouw@21153584
        name : "Leonie"
        parents : []>
      kind : <ML2_kind@18890040
        name : "Sjaakje"
        parents : []>>
    <ML2_ouders_van@25073696
```

```
      vader : <ML2_man@20112544
        name : "Piet"
        parents : []>
      moeder : <ML2_vrouw@21153584
        name : "Leonie"
        parents : []>
      kind : <ML2_kind@31575920
        name : "Petra"
        parents : []>>
    <ML2_ouders_van@31417672
      vader : <ML2_man@22436064
        name : "Sjaak"
        parents : []>
      moeder : <ML2_vrouw@18656248
        name : "Riet"
        parents : []>
      kind : <ML2_kind@27519608
        name : "Sjanet"
        parents : []>>}
  parents : []>

Backwards Output:
<ML1_bundel_tabellen@36236936
  ouders : Set{
    <ML1_moeder_kind_relatie@18784656
      elements : Set{
        <ML1_moeder_van@23848888
          from : <ML1_vrouw@23455720
            name : "Riet"
            parents : []>
          to : <ML1_kind@32595912
            name : "Pietje"
            parents : []>>
        <ML1_moeder_van@33323576
          from : <ML1_vrouw@36085192
            name : "Aukje"
            parents : []>
          to : <ML1_kind@29693640
            name : "Henkje"
            parents : []>>
        <ML1_moeder_van@25617896
          from : <ML1_vrouw@21345992
            name : "Leonie"
            parents : []>
          to : <ML1_kind@19695448
            name : "Sjaakje"
            parents : []>>
        <ML1_moeder_van@19902104
          from : <ML1_vrouw@21345992
            name : "Leonie"
            parents : []>
          to : <ML1_kind@35202368
            name : "Petra"
            parents : []>>
        <ML1_moeder_van@27519504
          from : <ML1_vrouw@23455720
            name : "Riet"
            parents : []>
          to : <ML1_kind@31339344
```

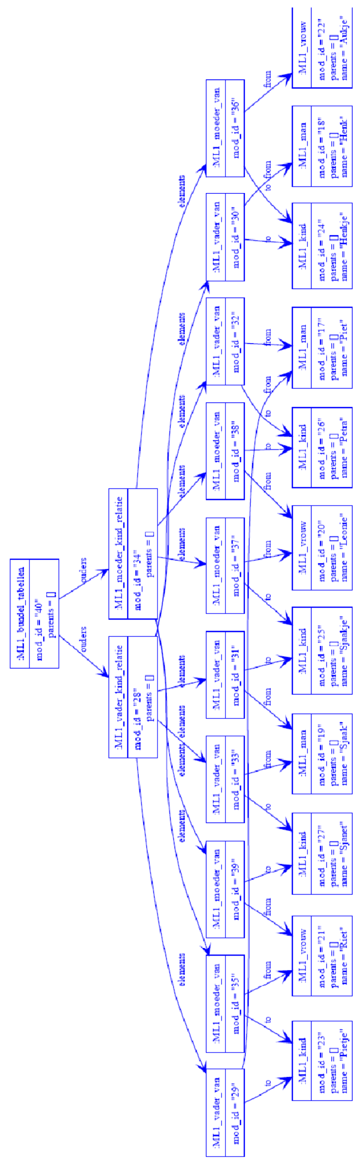
```
            name : "Sjanet"
            parents : []>>}
      parents : []>
    <ML1_vader_kind_relatie@38545136
      elements : Set{
        <ML1_vader_van@22102848
          from : <ML1_man@21228944
            name : "Piet"
            parents : []>
          to : <ML1_kind@32595912
            name : "Pietje"
            parents : []>>
        <ML1_vader_van@19695984
          from : <ML1_man@20112256
            name : "Henk"
            parents : []>
          to : <ML1_kind@29693640
            name : "Henkje"
            parents : []>>
        <ML1_vader_van@27520408
          from : <ML1_man@36276528
            name : "Sjaak"
            parents : []>
          to : <ML1_kind@19695448
            name : "Sjaakje"
            parents : []>>
        <ML1_vader_van@38545184
          from : <ML1_man@21228944
            name : "Piet"
            parents : []>
          to : <ML1_kind@35202368
            name : "Petra"
            parents : []>>
        <ML1_vader_van@21213896
          from : <ML1_man@36276528
            name : "Sjaak"
            parents : []>
          to : <ML1_kind@31339344
            name : "Sjanet"
            parents : []>>}
      parents : []>}
  parents : []>
```

A.7. Transformatie aanroep

```
func main():

        man1 := ML1.ML1_man("Piet")
        man2 := ML1.ML1_man("Henk")
        man3 := ML1.ML1_man("Sjaak")

        vrouw1 := ML1.ML1_vrouw("Leonie")
        vrouw2 := ML1.ML1_vrouw("Riet")
        vrouw3 := ML1.ML1_vrouw("Aukje")

        kind1 := ML1.ML1_kind("Pietje")
        kind2 := ML1.ML1_kind("Henkje")
        kind3 := ML1.ML1_kind("Sjaakje")
        kind4 := ML1.ML1_kind("Petra")
        kind5 := ML1.ML1_kind("Sjanet")

        vaders := ML1.ML1_vader_kind_relatie()
        vaders.elements.add(ML1.ML1_vader_van(man1, kind1))
        vaders.elements.add(ML1.ML1_vader_van(man2, kind2))
        vaders.elements.add(ML1.ML1_vader_van(man3, kind3))
        vaders.elements.add(ML1.ML1_vader_van(man1, kind4))
        vaders.elements.add(ML1.ML1_vader_van(man3, kind5))

        moeders := ML1.ML1_moeder_kind_relatie()
        moeders.elements.add(ML1.ML1_moeder_van(vrouw2, kind1))
        moeders.elements.add(ML1.ML1_moeder_van(vrouw3, kind2))
        moeders.elements.add(ML1.ML1_moeder_van(vrouw1, kind3))
        moeders.elements.add(ML1.ML1_moeder_van(vrouw1, kind4))
        moeders.elements.add(ML1.ML1_moeder_van(vrouw2, kind5))

        oudertabellen := ML1.ML1_bundel_tabellen()
        oudertabellen.ouders.add(vaders)
        oudertabellen.ouders.add(moeders)

        Sys.println("Input:\n", oudertabellen.to_str(), "\n\n")

        Sys.println("Output:\n")
        transformation := ML1_to_ML2(oudertabellen)
        transformed := transformation.get_target()
        Sys.println(transformed[0])

        Sys.println("\n\nBackwards Output:\n")

        transformationback := ML2_to_ML1(transformed[0])
        transformedback := transformationback.get_target()

        Sys.println(transformedback[0])
```

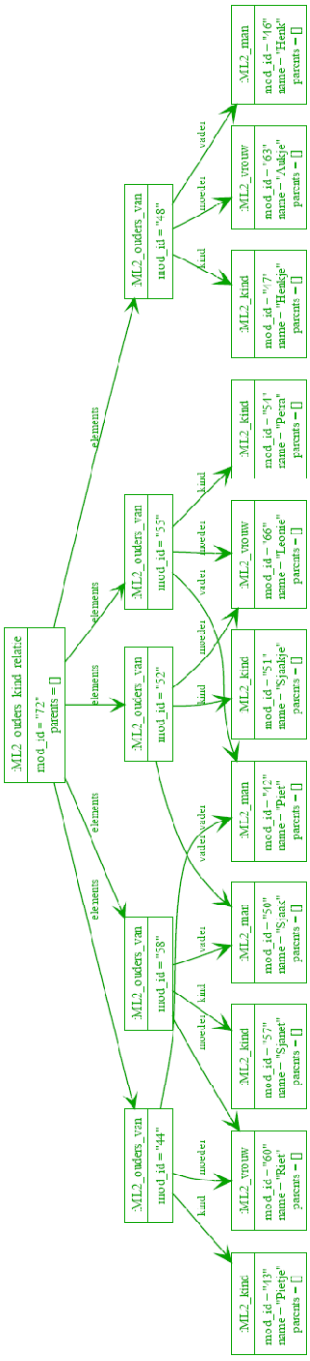
B. Gegenereerde MT visualisaties

B.1. De bronpopulatie



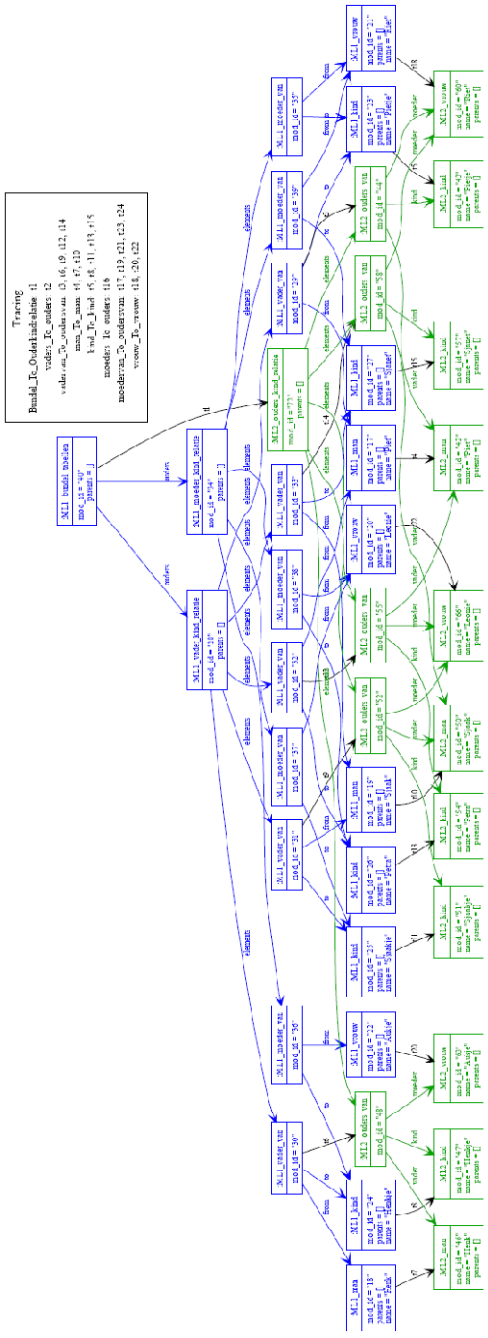
Figuur 11 – Bronpopulatie.

B.2. De doelpopulatie



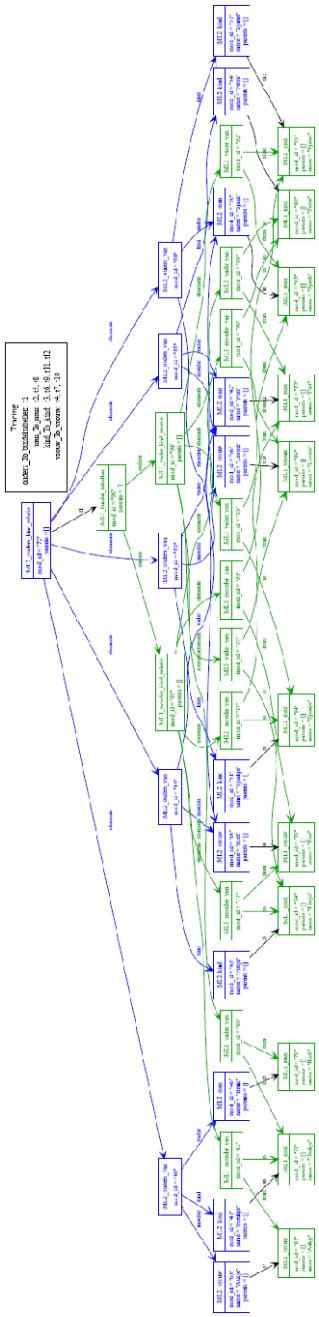
Figuur 12 – Doelpopulatie.

B.3. De traceinformatie



Figuur 13 – Traceinformatie heentransformatie.

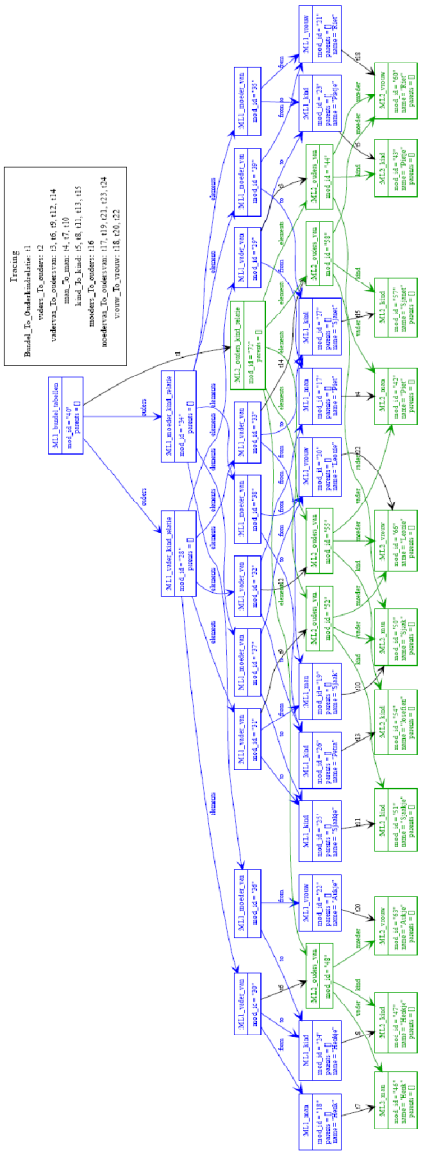
B.4. De traceinformatie van de transformatie terug



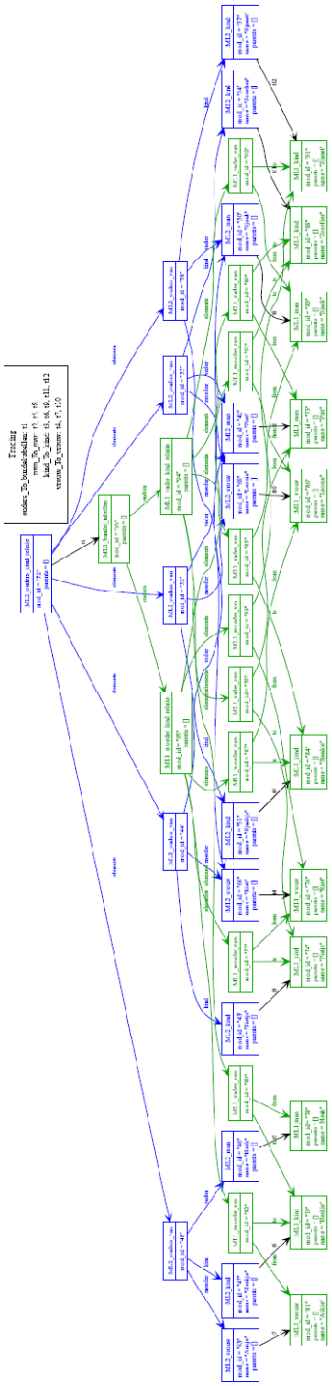
Figuur 14 – Traceinformatie terugtransformatie.

B.5. De traceinformatie van voorbeeld 2.3.1

Het voorbeeld in hoofdstuk 2 gaat nog niet om de concrete transformatie, maar om het laten zien van een praktisch nut van transformaties. Maar voor de volledigheid plaats ik hier de transformatie traces van het voorbeeld.



Figuur 15 – De voorbeeldtransformatie heen.



Figuur 16 – De voorbeeldtransformatie terug.