HOGESCHOOL GENT
LID VAN DE ASSOCIATIE UNIVERSITEIT GENT

Geassocieerde Faculteit Toegepaste Ingenieurswetenschappen
Valentin Vaerwyckweg 1, 9000 Gent

Academiejaar 2011-2012

Een relationele interface voor The DataTank

Masterproef voorgedragen tot het behalen van het diploma van
MASTER IN DE INDUSTRIËLE WETENSCHAPPEN: INFORMATICA

Pieter Colpaert
Promotoren: *Intern:* *Leen Brouns*
Tim De Pauw
Extern: *Bart Van Loon*

Samenvatting

Het doel van deze masterproef is de ontwikkeling van een module op *The DataTank*, teneinde ontwikkelaars toe te laten geschiktere uitvoer voor hun toepassing te verkrijgen uit meerdere databronnen door middel van een enkele oproep.

Eerst en vooral wordt aan de hand van een literatuurstudie bestudeerd wat *open data*, de rode draad doorheen de scriptie, inhoudt. Na een bespreking van de huidige legislatuur rond open data en het auteursrecht volgt een bespreking van de organisatie van een opendataevenement *Apps for Ghent*. Het tweede stuk van de literatuurstudie bespreekt het relationeel model, geeft uitleg over semantiek en leidt het *Semantische Web* in.

Vervolgens wordt er toegezien op de architectuur van The DataTank. The DataTank is een project geschreven in PHP dat dienst doet als data-adapter. Online beschikbare bronnen worden beschikbaar gesteld in webformaten, er worden statistieken vergaard op het gebruik en er kan feedback op stukken data gegeven worden. De architectuur van deze data-adapter wordt besproken om erna een relationele query-taal te implementeren, *TDTQL* gedoopt.

Tenslotte wordt de performantie van de relationele interface getest. De resultaten daarvan blijken acceptabel en geven een dieper inzicht in de werking van The DataTank.

Sleutelwoorden: open data; linked open data; Web Of Things; The Semantic Web; Web 3.0; webtechnologieën.

Abstract

The purpose of this applied dissertation is to develop a module upon *The DataTank* in order to allow developers to work with more appropriately structured information from multiple data sources through a single call.

Before all else, a literature study will research the meaning of *open data*, the leitmotif throughout this dissertation. Besides discussing current legislation on open data and copyright, the organisation of *Apps for Ghent*, an open data event, will be discussed. The second part of the literature study focuses on the relational model, explains semantics and gives an introduction to the *Semantic Web*.

Subsequently the architecture of The DataTank will be monitored. The DataTank is a project written in PHP that serves as a data adapter. Online sources will be made public in web formats, usage statistics will be gathered and user feedback will be made possible. The architecture of this data adapter will be discussed, to continue with the implementation of a relational query language, named *TDTQL*.

Finally the performance of this relational interface to The DataTank will be tested. The results are favourable and give a closer insight into how The DataTank works.

Keywords: open data; linked open data; Web Of Things; The Semantic Web; Web 3.0; webtechnologies.

Inhoudsopgave

Woord vooraf **2**

Inleiding **3**

i Twee vaststellingen . . . 3

ii Betrokken bedrijven en organisaties . . . 4

i Zeropoint.IT . . . 4

ii Het IBBT en het MultiMediaLab . . . 5

iii De iRail vzw . . . 5

I Rationale **6**

1 Open data **7**

1.1 Definitie . . . 7

1.2 Het auteursrecht . . . 8

1.2.1 Kritiek . . . 8

1.2.2 Een verbetering . . . 9

1.2.3 Licenties voor open data . . . 11

1.3 Open data in Europa . . . 11

1.3.1 EPSIplatform.eu . . . 11

1.3.2 Open Knowledge Foundation . . . 11

1.4 Een opendatabeleid . . . 12

1.4.1 Oneindige bron voor ontwikkelaars . . . 12

1.4.2 Maatschappelijke relevantie . . . 12

1.4.3 Vrije markt . . . 12

1.4.4 Een economische meerwaarde . . . 12

1.4.5 Business models . . . 13

1.5 Open Government Data Camp 2011 . . . 13

2 Apps for Ghent **16**
2.1 De Stad Gent 16
2.2 Een event 17
2.3 Organisatie 17
2.3.1 Outerthought 18
2.3.2 Xplore+ 18
2.4 Apps 18
2.5 Feedback en nabespreking 19
2.6 Conclusie 20

3 Het relationele model **21**
3.1 Relaties 21
3.2 Een relationele interface 21
3.2.1 REST 22
3.2.2 HTSQL 22
3.3 Relationele databanken in vraag gesteld 23
3.3.1 CAP ou pas CAP? 23
3.3.2 100% partitietolerantie 24
3.3.3 Datarepresentatie 25

4 W3C en het Semantische Web **26**
4.1 Semantiek en the Web Of Things 26
4.2 Mark-up- en querytalen 28
4.2.1 Semantische formaten 28
4.2.2 SPARQL 33
4.3 Semantiek, REST en HTTP Range 14 issue 34
4.3.1 REST 34
4.3.2 Het Semantische Web en REST 35
4.3.3 HTTP Range 14 35
4.4 Het Semantische Web vandaag en The DataTank 36

II Analyse en implementatie **39**

5 The DataTank **40**
5.1 Controle op data 40
5.2 Geen dubbel werk: webformaten 42
5.2.1 Installed resources 42
5.2.2 Generic resources 42

5.2.3 Remote resources . . . 43
5.3 URL-structuur . . . 43
5.3.1 Package/resource-paar . . . 43
5.3.2 Verplichte parameters . . . 44
5.3.3 REST-filters . . . 44
5.3.4 Open Search . . . 45
5.4 Documentatie voor ontwikkelaars . . . 45
5.4.1 Resources . . . 45
5.4.2 Admin . . . 46
5.4.3 Queries . . . 46
5.4.4 Ontology . . . 46
5.4.5 Formatters . . . 46
5.4.6 Exceptions . . . 46
5.5 Een tweerichtingsverkeer . . . 46

6 Architectuur van The DataTank 48
6.1 Controller . . . 48
6.1.1 RController . . . 49
6.1.2 CUDController . . . 50
6.2 Views . . . 50
6.3 Model . . . 50
6.4 Eigen resources schrijven . . . 51
6.5 Eigen strategies schrijven . . . 51

7 De relationele interface 53
7.1 Waarom een simpele relationele interface? . . . 53
7.2 TDTQL . . . 54
7.2.1 Een endpoint . . . 54
7.2.2 Relationeel: best-effort . . . 55
7.2.3 De basisfuncties . . . 55
7.3 De parser . . . 56
7.3.1 Backus-Naur Formaat . . . 56
7.3.2 Implementatie . . . 57
7.4 Integratie in het project . . . 63

III Evaluatie 64

8 Performantie-analyse 65

8.1 TDTQL vs. REST 66
8.1.1 Opstelling 66
8.1.2 Het resultaat 66
8.1.3 Verklaring van het resultaat 67
8.1.4 Conclusies 67
8.2 Benchmark van uitgebreide queries 68
8.2.1 Selecteren 68
8.2.2 Sorteren 68
8.2.3 Filteren 70
8.2.4 Linken 70
8.2.5 Conclusie 71

9 Gebruik en toekomst 72

9.1 Gebruik 72
9.1.1 data.iRail.be 72
9.1.2 Apps for Ghent 2 73
9.1.3 Wenen 74
9.1.4 Zeropoint.IT 74
9.2 TDTQL nu en morgen 74
9.2.1 Beperkingen 74
9.2.2 Volgende stappen 75

Conclusie 77

Woordenlijst 82

Woord vooraf

Het was een boeiend jaar. Ik ben met de meest interessante mensen in contact gekomen. Ik ben naar Polen mogen reizen om een presentatie te geven rond open data [1]. Met de opgebouwde contacten wordt binnenkort een nieuwe vereniging opgericht rond vrije kennis in België (The Open Knowledge Foundation Belgium). Samen met volksvertegenwoordiger Peter Dedecker en projectleider bij het Instituut voor BreedBandTechnologie (IBBT) dr. Erik Mannens hebben we *Apps For Ghent* georganiseerd. Dit zou niet mogelijk zijn geweest zonder bepaalde mensen, waarvoor een woord van dank hier niet misplaatst is.

Eerst en vooral wil ik een grote bedanking richten aan mijn externe promotor Bart Van Loon voor zijn oneindig geduld en blijvende steun in heel drukke tijden. Ook m'n interne promotoren, Leen Brouns, Tim De Pauw en tweede lezer Wijnand Schepens ben ik dank verschuldigd om me op het goede pad te wijzen en te houden.

Bedankt aan dr. Erik Mannens die vanaf september meer dan een volwaardig promotor heeft betekend en aan zijn teamleden, onderzoeker Miel Vander Sande en dr. Davy Van Deursen, die over de architectuur nooit een discussie uit de weg zijn gegaan. Bart Rosseau, ambtenaar 2.0 bij de Stad Gent, bedank ik hier ook even voor een blik binnen het trage, of perfectionistische, beleid van een lokale overheid, voor de vele open discussies waartoe hij me uitnodigde en de lift naar het "Leuven Open Data"-evenement.

In deze periode is ook de *iRail vzw*, waar ik sedert de maand voor aanvang van de masterproef ondervoorzitter van ben, heel erg gegroeid. Bedankt aan hen voor het eindeloos vertrouwen om *The DataTank* te adopteren als het opensource-vlaggenschipproduct, voor het verzorgen van de legale kant en voor de vele enthousiastelingen die al in een vroeg stadium met The DataTank aan de slag gingen. Zij lieten toe de architectuur aan de realiteit te toetsen. Een speciale bedanking gaat daarbij uit naar Yeri Tiete, Lieven Janssen, Jan Vansteenlandt, Tim Besard, Dimitri Roose, Quentin Kaiser, Christophe Versieux en onze advocaat Ywein Van den Brande.

Als laatste, en daarom zeker niet minder belangrijk, bedank ik mijn ouders, broers, zussen en vrienden voor hun steun, eindeloos geduld, het verdragen van m'n humeur als ik weerom tot de latere uren gewroet had op een probleem en voor de inspiratie die vaak op onverwachte momenten kwam dagen.

Inleiding

“

Data is a precious thing and will last longer than the systems themselves.

Tim Berners-Lee,
directeur van het World Wide Web consortium
en het Open Data Institute

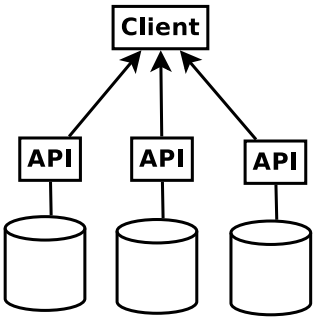
”

In dit hoofdstuk wordt eerst de huidige situatie geschetst aan de hand van enkele vaststellingen. Daarna worden de betrokken organisaties voorgesteld.

i Twee vaststellingen

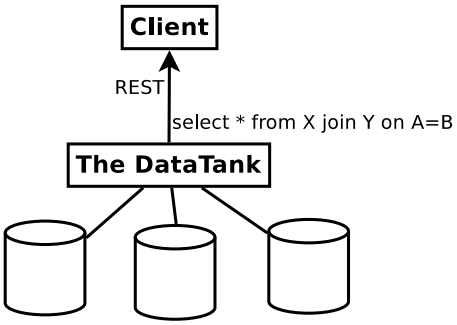
Het project *The DataTank* is gebaseerd op twee vaststellingen. De eerste vaststelling is dat een programmeur van een applicatie multigetalenteerd moet zijn en verschillende omgevingen moet kunnen beheersen om een toepassing te bouwen. Niet alleen moet hij een expert zijn in het programmeren en ontwerpen van een grafische schil, hij moet ook bedreven zijn in het schrijven van een datalaag. Deze datalaag zit vaak op de server zelf. Dat aan deze datalaag jammer genoeg vaak het minste aandacht wordt besteed, mochten verschillende projecten al in het verleden bekopen met slechte publiciteit. Hierbij verwijzen we naar veelgebruikte apps waar security issues snel werden blootgelegd[2]. Ook grote Belgische bedrijven als Telenet blijken kwetsbaar voor dergelijke euvels[3].

Een tweede vaststelling is dat een ontwikkelaar heel vaak geconfronteerd wordt met online datalagen, verpakt in een Application Programming Interface (API). Zo zal hij niet in een vraag alle benodigde data kunnen ophalen. Aangezien de app developer nu niet meer zelf de datalaag programmeert, zal de ontwikkelaar de applicatie zelf te veel requests laten doen. Nu is het bijvoorbeeld ondenkbaar dat een mobiel toestel zelf een intermodale route zal zoeken om van punt A naar punt B te gaan: er zouden te veel (online) databronnen geraadpleegd



Figuur 1: Onhoudbaar model voor mobiele toepassingen: de client moet te veel data binnenhalen

moeten worden. Al deze databronnen moeten eerst over het mobiele netwerk gedownload worden om vervolgens verwerkt te worden.



Figuur 2: De toekomstige architectuur van The DataTank

Het uiteindelijke doel van deze masterproef is de ontwikkeling van een module op The DataTank, teneinde ontwikkelaars toe te laten geschiktere uitvoer voor hun toepassing te verkrijgen uit meerdere databronnen door middel van een enkele oproep.

ii Betrokken bedrijven en organisaties

i Zeropoint.IT

Zeropoint.IT is een gemanaged offshoringsbedrijf. Het heeft heel wat talent samengebracht in Islamabad (Pakistan). Vanuit België regelt Bart Van Loon en zijn partner ontwikkelaars uit Pakistan voor jouw project.

In 2008 richtte Bart Van Loon met enkele partners het project *The DataTank* op. Het richtte zijn pijlen op overheidsbedrijven die voor derden nuttige metadata of gegevens in een relationele databank opslaan. De markt bleek nog niet

klaar en in het najaar van 2009 werd het project opgedoekt, samen met de val van de toenmalige regering.

ii Het IBBT en het MultiMediaLab

Het IBBT is een vereniging zonder winstoogmerk die door innovatie in ICT een blijvende en positieve impact wil creëren op de maatschappij.

Het MultiMediaLab is een onderzoeksgroep aan het IBBT. Een van de onderzoeksonderwerpen is metadatatechnologie. Bij het MultiMediaLab loopt nu een onderzoek naar het Semantisch Web (zie hoofdstuk 4), waar The DataTank een gelijkaardig doel voor ogen houdt. De iRail vzw (zie vervolg) en het MultiMediaLab werken vanaf september 2011 samen rond het Semantisch Web.

iii De iRail vzw

iRail werd gestart in 2008 als een mobiele website die vandaag nog steeds online beschikbaar is. Yeri Tiete ("Tuinslak") nam vaak de trein en bij het aanschaffen van zijn iPhone wou hij een mobiele website, toen nog onbestaande voor de Belgische spoorwegen, zodat hij makkelijk zijn route zou kunnen plannen.

Na heel wat media-aandacht [4] werd in september de iRail API gelanceerd. Deze API, een interface voor programmeurs om zelf applicaties met deze data omtrent openbaar vervoer te bouwen, werd al snel de motor van een rijke community.

In januari 2011 werd de iRail vzw opgericht. iRail is niet alleen een mobiele website meer, maar het is een consortium geworden om te werken naar een betere opentransportdatabeleid.

De iRail vzw is een living lab, of proeftuin, voor ideeën en projecten rond open data[1]. Als living lab steunt iRail projecten met hosting, support op datasets, design, visibiliteit en hulp bij het beschermen van intellectueel eigendom.

[1] Op de definitie van open data wordt er in hoofdstuk 1 dieper ingegaan.

Deel I

Rationale

Hoofdstuk 1

Open data

“

If I had asked people what they wanted,
they would have said faster horses.

Henry Ford,
industrieel pionier
en oprichter van Ford Motor Company

”

Open data is de rode draad doorheen deze scriptie. Niet alleen op academisch vlak is het interessant, ook vanuit economisch perspectief biedt het opportuniteiten. The DataTank is een platform dat zich hier volledig op toelegt. Maar wat zijn open data nu net?

1.1 Definitie

Open data zijn gegevens die vrij ter beschikking worden gesteld, zonder restricties opgelegd door het auteursrecht, patenten (op bijvoorbeeld de drager), of andere controlemechanismen.

De vijf sterren van open data Er zijn verschillende gradaties van open data, niet alleen op het vlak van openheid, maar ook op het vlak van kwaliteit van de datasets. Tim Berners-Lee [5] stelde de vijf sterren van open data voor. De eerste ster is wanneer data online beschikbaar worden gemaakt zonder dat deze snel geïnterpreteerd kunnen worden door een machine. Hoe meer sterren, hoe kwaliteitsvoller. De vijfde ster is wanneer data online beschikbaar worden gemaakt, interpreteerbaar zijn door machines en gelinkt worden aan andere datasets.

1. Maak data online beschikbaar op het Web (eender welk formaat)

2. Maak het beschikbaar als gestructureerde data (bv. excel-file in de plaats van een image scan),

3. Geen proprietair formaat (bv. JSON/XML/CSV in plaats van XLS),

4. Gebruik Uniform Resource Identifier (URI)'s om *dingen* te identificeren, zodat derden ernaar kunnen linken en

5. Link data aan andere mensen hun data en voorzie context

Ster een tot drie kunnen worden samengevat als data ter beschikking stellen in een web-service. Bij ster 4 worden URI's gebruikt om de data te identificeren. Een mogelijkheid waar in hoofdstuk 3.2.1 en 4.3.1 dieper op ingegaan wordt.

Zonder restricties De meeste[1] datasets zijn volgens de Europese wetgeving onderworpen aan het auteursrecht en daar kan geen afstand van genomen worden zonder licenties. Vaak vallen datasets niet onder de noemer "open data" omdat de gekozen licenties te restrictief zijn, of omdat er gewoon helemaal nooit gedacht is aan een licentie. In dit laatste geval heeft een gebruiker helemaal geen rechten. Ook voor open data zijn er verschillende licenties. Deze worden besproken in hoofdstuk 1.2.3.

1.2 Het auteursrecht

In de jaren 1980 werd over heel de wereld het auteursrecht hervormd en gelijkgesteld. Het meest ingrijpende dat toen werd beslist is dat al het werk dat gecreëerd zou worden, automatisch "all rights reserved/alle rechten voorbehouden" is. Voordien moest een creatief werk expliciet worden geregistreerd indien alle rechten moesten voorbehouden worden. Schrijft een persoon iets, dan is hij of zij de enige eigenaar en dit tot minstens 70 jaar na de dood. Niemand mag iets doen met dit werk zonder uitdrukkelijke toestemming. Om iemand anders toch rechten te geven op het werk (om het te publiceren, om het te hergebruiken, om het te herwerken, enz.) kan dat via schriftelijke toestemming.

1.2.1 Kritiek

Een artiest of een auteur is evengoed een ondernemer die vergoed moet worden voor zijn werk. Uiteraard mag dit niet ten koste van innovatie gaan en mag dit systeem niet misbruikt worden voor andere doeleinden.

[1] Uitzonderingen vallen onder het 2e artikel van de Europese databankwet. Deze datasets zijn niet auteursrechterlijk beschermd als ze een logisch gevolg zijn van een bepaalde activiteit. Zo zijn de resultaten van een paardenkoers of loterij niet auteursrechterlijk beschermd.

De permissie-cultuur Vanaf de jaren 1990 was ook het internet in volle opmars. Het werd steeds makkelijker voor derden om materiaal te hergebruiken. Voor velen was dit heuse innovatie: iemand kon bijvoorbeeld een foto van een werk, die een ogenblik geleden genomen was, aan de andere kant van de wereld onmiddellijk hergebruiken in een presentatie. Voor hen is innovatie de mogelijkheid om voort te werken op eerder gedaan werk, en hun werk weerom beschikbaar te maken voor hergebruik.

Nu deze wetgeving werd ingevoerd, kunnen werken niet zomaar worden hergebruikt. Er moet namelijk bij ieder werk, ook al werd dat niet expliciet beschermd door de auteur of artiest, een schriftelijke toestemming worden gevraagd aan hem of zijn nabestaanden. Dit zorgde voor een drastische vertraging, want voor alles dat zou worden hergebruikt, moest nu gewacht worden op een schriftelijke bevestiging.

De duur Een werk blijft eigendom van een persoon tot minstens 70 jaar na zijn of haar dood. Dat dit te lang is, kan enkel worden gemerkt aan werk waarvan de auteur al even overleden is. Zo wordt het zingen van het liedje "Happy Birthday" aanzien als schending van het auteursrecht. Voor meer informatie verwijzen we naar Brauneis [6].

Fair use (eerlijk gebruik) Eerlijk gebruik is een te weinig gekende clausule in de auteursrechtenwetgeving. "Fair use" is helaas te weinig duidelijk gedefinieerd en het is aan elke natie om er zelf een betekenis aan te geven. In de loop der tijd is er echter nog heel weinig fair use. De lijn wordt heel dun: zo is een cd uitlenen aan een vriend fair use, maar een album doorsturen via het internet een zwaar misdrijf.

Niet alleen wordt "fair use" door tussenpersonen geminimaliseerd, het delen op zich wordt gecriminaliseerd. Het doorgeven van een werk wordt door sommigen "stelen" genoemd en heuse campagnes worden opgezet om mensen schrik aan te jagen.[2]

1.2.2 Een verbetering

Schriftelijke toestemmingen werden door advocaten omgevormd tot algemene licenties of "general licenses". Deze geven alle mensen meer rechten op een bepaald werk.

FLOSS: Free, Libre, Open Source Software Nadat het internationale auteursrecht in de jaren 1880 hervormd werd, deed dat in de software-wereld heel wat stof opwaaien. Als reactie op het feit dat elk creatief werk auteursrechterlijk beschermd is, kwamen er de algemene licenties. Het was immers, althans in Europa, niet meer toegestaan om te zeggen: "mijn werk is vrijgesteld van het auteursrecht". Algemene licenties (of general licenses) zijn op de dag van vandaag een manier om iedereen meer vrijheid te geven (minder vrijheid kan immers niet) om werk te gebruiken. Tussen verschillende instanties, voornamelijk

[2] Cf. de "Piraterij is een misdrijf"-campagne.

het Massachusetts Institute of Technology (MIT), de University of California Berkeley (UCB) en de Free Software Foundation (FSF) (onder leiding van Richard Stallman) werd bediscussieerd wat, in de context van software, "vrijheid" inhield. De welbekende discussie "is vrijheid zo vrij dat je iemand anders zijn of haar vrijheid kan afnemen?" was een feit.

Creative Commons Pas in het begin van de 21e eeuw werd de vereniging *Creative Commons*(CC) opgericht. De Creative Commons-licenties zijn evenwel een collectie van algemene licenties die zich niet op software richten, maar op creatieve werken zoals boeken, foto's, bewegende beelden, muziek ... Ze werken met clausules die kunnen worden toegevoegd: mijn werk mag geremixt, hergebruikt en gedeeld worden op voorwaarde dat ...

1. CC0 (zero): geen voorwaarden.
2. BY (Attribution): de naam van de auteur moet vermeld worden. Dit kan door een link terug te plaatsen bij het afgeleide of gebruikte werk of de naam te vermelden.
3. SA (Share Alike): het werk op dezelfde manier gelicentieerd wordt. Dit zorgt ervoor dat ook afgeleid werk telkens vrij beschikbaar blijft.
4. NC (Non Commercial): het werk niet voor commerciële doeleinden wordt gebruikt. Commerciële doeleinden moeten echter nog verder gespecificeerd worden door de gebruiker.
5. ND (Non Derivatives): er geen veranderingen aan het werk zelf worden gedaan.

Deze clausules zijn een gebruiksvriendelijke methode om in mensentaal uit te leggen wat er met creatief werk mag gebeuren. Als een dergelijke licentie gebruikt wordt, kan automatisch een tekst in juridische termen die al getest werd in rechtbanken geraadpleegd worden.

Van de mogelijke combinaties clausules zijn er maar enkele die vallen onder "vrije cultuur": CC0, CC BY en CC BY SA. Dit zijn respectievelijk een licentie die alle rechten toelaat aan derden, een licentie die verwacht dat bij hergebruik de naam van de originele auteur behouden blijft en een licentie die ook verwacht dat het afgeleide werk onder dezelfde licentie naar buiten gebracht wordt.

Ook de meest gebruikte algemene softwarelicenties kunnen onderverdeeld worden zoals Creative Commons-licenties: de MIT en Berkeley Software Distribution (BSD) (van UCB) licenties die toelaten dat code wordt hergebruikt mits naamvermelding en de GNU General Public License (GPL)-familie die toelaat dat code wordt hergebruikt mits de naam wordt vermeld van de oorspronkelijke auteur en de afgeleide code onder dezelfde licentie wordt uitgebracht.

Patenten Er kan ook nog worden opgemerkt dat in sommige landen computerprogramma's ook worden afgeschermd door middel van een patent. Aangezien dit in België maar in mindere mate mogelijk is[3] en wegens de omstreden

[3]Door het Europese Patent Treaty artikel 52 kan software niet gepatenteerd worden. Artikel 52.3 nuanceert dit dan weer door een "als dusdanig" clausule toe te voegen. Een ongeschreven

aard gaan we daar niet dieper op in. Het auteursrecht zorgt immers al voor de bescherming van software en data. Naast computerprogramma's zijn ook recepten en wiskundige formules, in tegenstelling tot de Verenigde Staten, niet patenteerbaar in Europa.

1.2.3 Licenties voor open data

Ook datasets zijn auteursrechterlijk beschermd, meer bepaald door de databankwet. Voor deze datasets zijn ook specifieke opendatalicenties gemaakt zoals de EUPL (European Public License). Ook de standaard Creative Commonslicenties worden gebruikt voor datasets. Er kan in dat geval slechts van open data gesproken worden als er niet meer dan de vlaggen BY en SA worden gebruikt.

Hoewel vaak gehanteerd voor data, zijn de CC-licenties zijn niet specifiek opgesteld voor gegevens. Hier biedt the Open Cata Commons een oplossing. Deze vereniging heeft als enig doel licenties op te stellen om open data juridisch te definiëren. Hun meest gebruikte licentie is op het moment van schrijven de Open Data Commons Open Database License (ODbL). Ze hanteren gelijkaardige vlaggen als Creative Commons.

1.3 Open data in Europa

1.3.1 EPSIplatform.eu

Het European Public Sector Information (EPSI)-platform is een Europese spreekbuis voor specialisten inzake het hergebruik van informatie van openbare instellingen. Sinds 2008 is er een Europese richtlijn[4] van kracht die ijvert voor meer transparantie en open data. De website biedt voor openbare instellingen een portaal aan met Europese richtlijnen voor een beter open data beleid. Op de webpagina kan eender wie terecht voor algemene informatie over EPSI (een oudere benaming voor open data), nieuws, blogs, enzovoort.

1.3.2 Open Knowledge Foundation

De OKFN is een vereniging die aanvankelijk begon in het Verenigd Koninkrijk. Door events te organiseren wordt binnen de OKFN een heel levendige community onderhouden. De vereniging organiseerde bijvoorbeeld ook dit jaar het *Open Government Data Camp* en hebben tal van projecten op hun naam. Een van de meest gekende projecten is de *open data manual* voor ambtenaren.

---

en heel vaak bekritiseerde interpretatie is dat software in Europa niet patenteerbaar is zolang ze geen technische bijwerking heeft. Zo werd in Duitsland wel het patent voor *VFAT* aanvaard en zijn er heel wat softwarepatenten in het Verenigd Koninkrijk.

[4] `http://ec.europa.eu/information_society/policy/psi/rules/ms/index_en.htm` - European Public Sector Information (EPSI) directive

1.4 Een opendatabeleid

1.4.1 Oneindige bron voor ontwikkelaars

Er zijn heel wat mobiele en vaste toestellen in omloop. Als bedrijf zorgen dat klanten zo goed mogelijk geïnformeerd worden door op alle toestellen zelf een toepassing uit te brengen is onbegonnen werk. Als ruwe data beschikbaar gesteld worden, kunnen ontwikkelaars zelf voor hun toestel een client bouwen indien die niet aanwezig is.

Het is mogelijk dat verschillende datasets verwerkt zullen worden in een mash-up. Een mash-up kan nieuwe relaties bloot leggen die anders niet mogelijk waren.

Hoe meer data er dus beschikbaar zijn, hoe groter de kans dat er een betere berichtgeving zal zijn. Men zegt ook wel eens dat open data de serendipiteit van het Web verhoogt.

1.4.2 Maatschappelijke relevantie

Een sterk argument voor open government data is dat de burgers er al voor hebben betaald via belastingen. Moeten die dan nog eens betalen bij het willen bekijken van de data?

Het is van groot belang voor de democratie dat data van overheden worden ontsloten. Het is de manier waarop burgers hun overheid kunnen controleren. Ook bepaalde data van bedrijven kunnen belangrijk zijn in het kader van openbaarheid van bestuur.

1.4.3 Vrije markt

Bij het openen van data zullen er meer clients gebouwd worden voor meer platformen dan je ooit zelf had kunnen opleveren. Er wordt een vrije markt gecreëerd rondom een bepaalde dataset. Hoe beter een app op een bepaald platform, hoe meer die zal gedownload worden.

Meer datasets die op een creatieve manier kunnen gebruikt worden door derden leiden zo onrechtstreeks tot meer tewerkstelling.

1.4.4 Een economische meerwaarde

Er moet een onderscheid gemaakt worden tussen *statische*, *dynamische* en *real-time* data. Statische data, zoals historische gegevens of statistieken, veranderen nooit. Dynamische data zijn data die af en toe kunnen veranderen (bv. haltes van de bussen) en geldig blijfen voor een bepaalde tijdspanne. Real-time data zijn data die zich over het nu uitspreken (bv. treinvertragingen) en heel snel veranderen.

Om een opendatabeleid in te voeren op dynamische of real-time data kunnen we niet zomaar de data te download zetten. Als datasets bij derden geraken, en als er wijzigingen op die datasets optreden, moet de data-eigenaar zeker kunnen

zijn dat overal de recentste data worden gebruikt, en niet de foutieve oude. Als reactie hierop zullen bedrijven ofwel afzien van een opendatabeleid, of een policy ontwikkelen waarbij er eerst een overeenkomst moet worden getekend of een key moet worden aangevraagd met een uitgebreide projectbeschrijving. Een betere oplossing, die we ook zullen aanreiken binnen de context van The DataTank, is dat data worden aangeboden via een service waar iedereen altijd vragen aan stelt. Dit levert extra voordelen op die vooral te maken hebben met de feedback van gebruikers.

Een opendatabeleid invoeren in een bedrijf of organisatie kan een directe meerwaarde betekenen, zeker als we het hebben over feedback op data.

1.4.5 Business models

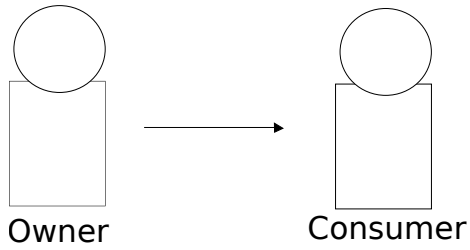
Een opendatabeleid, net als bij het concept open source, wordt ook vaak aangehaald in het kader van business models. Open data is een business model waar je een return on investment kan creëren door het ontsluiten van gegevens.

Enkele voorbeelden

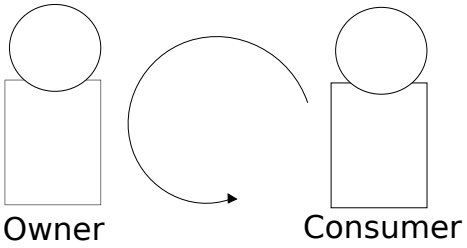
- Als een commerciële partij zeker wil zijn van snelle access tot dynamische data zonder dat andere spelers die access kunnen beïnvloeden op piekmomenten, kan er een *Quality Of Service (QoS)*-contract aangeboden worden.
- Data-eigenaars hebben de grootste kennis over de eigen data. Het is vanzelfsprekend dat zij deze data het beste kunnen aanleveren naar commerciële partijen onder eender welke vorm. Daarvoor kan betaald worden.
- Support en training kan geleverd worden om te leren werken met deze data.
- Version-management: het onderhouden van een community release en een stable release van de data. De community release is het meest up-to-date maar kan fouten bevatten. De stable release is niet helemaal up-to-date maar de data die erin zitten zijn gedubbelcheckt geweest door een manager. Community versies worden dan, net zoals Wikipedia, beheerd door de gebruikers zelf.

1.5 Open Government Data Camp 2011

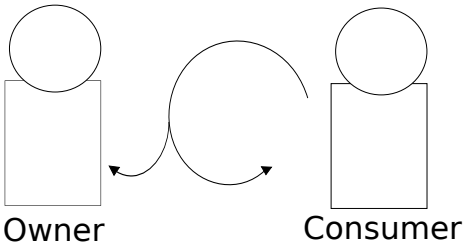
Op 20 en 21 oktober 2011 ging in Warschau het Open Government Data Camp door. Het team achter The DataTank was uitgenodigd om een talk te geven van 15 minuten. De titel van de talk was “follow the stars”. Daarin werd The DataTank gebruikt als voorbeeld om de vijf sterren van open data te gebruiken als stappenplan om datasets te ontsluiten. De slides van de talk kunnen online bekeken worden. [1]



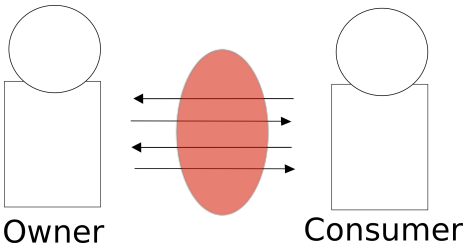
(a) Ter beschikking stellen: de meest voor de hand liggende vorm van open data



(b) Crowd sourcing: gebruikers verzamelen zelf data verzamelen en stellen ze ter beschikking van derden.



(c) Een community levert feedback. Dit kan gaan van simpele statistieken tot foutcorrecties.



(d) Als real-time data via een web-service opgehaald worden, kan dit verkeer geanalyseerd worden.

Figuur 1.1: Er zijn drie strategieën, al dan niet bewust, waarop een organisatie of bedrijf een meerwaarde kan halen uit datafeedback op dynamische en real-time data: crowd sourcing (fig. 1.1b), leren uit feedback (fig. 1.1c) en het analyseren van het gebruik (fig. 1.1d).

Figuur 1.2: Open Government Data Camp 2011: een stukje van de groepsfoto

Hoofdstuk 2

Apps for Ghent

Eerder werd al besproken wat het doel van deze masterproef inhoudt en wat open data zijn. Voor er wordt overgegaan op een uitgebreide literatuurstudie bespreken we in dit hoofdstuk de organisatie van een evenement om data-eigenaars in contact te brengen met data-consumenten.

2.1 De Stad Gent

Binnen de Stad Gent, zo beweert strategisch communicatie-manager van de Stad Gent Bart Rosseau, zijn ze al goed vertrouwd met de idee van open data. Er zijn helaas nog drie zaken die hierbij opgemerkt moeten worden.

Een eerste vaststelling is dat werken met open data een loutere beleidskeuze is. Dat wil zeggen dat het al dan niet openstellen van gegevens niet in handen ligt van informatici, maar van de kaderleden die hogerop de keuzes maken. Aangezien de logische economische stap lijkt om deze data te exploiteren door ze door te verkopen aan derden, wordt de optie om hun gegevens te ontsluiten vaak over het hoofd gezien. Eenmaal overtuigd dat open data zowel op economisch als technologisch vlak een betere beleidsbeslissing is, stellen er zich een hoop nieuwe vragen: wat hebben we daar allemaal voor nodig? Welke formaten moeten we gebruiken? Wat met privacy-gevoelige data? Wanneer moeten we nu de knoop doorhakken? Hoe zorgen we dat mensen van onze data gebruik maken?

Een tweede vaststelling is dat grote organisaties, bedrijven of overheden bekend staan om hun trage werking. De oorzaak zou kunnen gelegd worden bij het slecht functioneren van deze instanties, kafkaëske praktijken of incompetentie van werknemers, maar dat zou uiteraard veel te gemakkelijk zijn. We leggen de hoofdfactor liever bij perfectionisme. De Stad Gent wil voor haar inwoners, zoals ook de NMBS voor reizigers, alleen maar het beste. Dit geeft als gevolg dat ze ook alleen maar 100% perfecte datasets willen aanleveren en 100% perfecte software willen aanbieden. Aangezien dit per definitie – van software wordt beweerd dat ze nooit “af” is – nooit mogelijk is, kan dit uiteraard heel lang duren.

Als derde en laatste vaststelling is het voor sommige heel interessante data heel moeilijk om ze te ontsluiten. Vaak staan er bij wet auteursrechten op, maar blijkt niemand precies te weten wie de auteursrechten beheert of bij wie een licentie bekomen kan worden. Als dan toch de eigenaar bekend is, blijkt deze weer bij een andere instantie te zitten. Als ontwikkelaar bij dergelijke instanties langs gaan om verschillende datasets te verkrijgen voor een simpele toepassing is onbegonnen werk.

2.2 Een event

Op zaterdag 14 mei 2011 vond Apps For Ghent plaats in de Zuiderpoort te Gent. De dag verliep alsvolgt:

- 10:00 - Presentatie van volksvertegenwoordiger Peter Dedecker over open data binnen De Kamer
- 10:15 - Presentatie van Resul Tapmaz, Gentse schepen voor IT, over open data bij de Stad Gent
- 10:30 - Erik Mannens van het IBBT praat over het LODGE project
- 10:45 - Jens Dehaese spreekt over Ghent Web Valley
- 11:00 - Bart Rosseau stelt Gentse data voor die beschikbaar wordt
- 11:15 - Luk Verhelst stelt de Cultuurnet API voor: alle opkomende evenementen in Vlaanderen
- 11:30 - Bart Nelis praat over data bij De Lijn
- 11:45 - Marc Portier stelt Lily 1.0 voor
- 12:00 - Lunch en start van de hackathon
- Vanaf nu werken mensen aan hun idee in groepen van 2 tot 5 personen
- 17:00 - Voorstelling ideeën en prijsuitreiking

Dat Apps For Ghent een succes was, mag gezegd worden. Voor een volledig verslag verwijzen we naar De Standaard [7].

2.3 Organisatie

Op 12 maart, 2 maanden voordien, werd Apps For Amsterdam georganiseerd. Met een gelijkaardig programma en met hetzelfde doel was dit een enorme inspiratiebron. Hack De Overheid, een stichting die creatieve projecten doet rond digitalisering van overheden, bracht op die dag een honderdtal mensen samen om rond open data een open discussie te houden.

Apps For Ghent, aanvankelijk puur een idee vanuit het perspectief van deze masterproef, kon niet door een persoon tot stand gebracht worden. Er waren

datasets nodig, mensen die met data aan de slag gingen, mensen die kwamen spreken over dataverwerking, enzovoort. De eerste persoon die hierrond gecontacteerd werd, was Peter Dedecker, volksvertegenwoordiger in de Kamer. Dat hij de ideale persoon was om dit evenement te trekken, bewijst hij meermaals. Om toegang tot data als technocratisch en maatschappelijk belangrijk thema ook in de Kamer aan bod te laten komen, richtte hij `peteropentdata.be` op. Via deze site kunnen datasets die in de toekomst toegankelijker moeten zijn, worden aangevraagd. Ook wist Peter Dedecker, die vóór zijn functie als volksvertegenwoordiger werkte in het IBBT, Erik Mannens, projectmanager bij het IBBT en een van de mensen achter het toekomstige Linked Open Governmental Data Environment (LODGE)-project, aan te duiden als medeorganisator. Ook Bart Rosseau, strategisch communicatie-manager bij de Stad Gent, wist hij te contacteren als de man binnen het Gentse stadsbestuur. Als sponsors, buiten de locatie en accommodatie die geleverd werden door het IBBT, boden zich Outerthought en Xplore+ aan.

Bart Rosseau zou kijken om data van binnen de Stad Gent mee te brengen. Hieruit kwamen vooral geografische data, 3D-data van de Nederschelde en Gent in cijfers (statistieken). Via de iRail vzw werd ook de online manager bij De Lijn Bart Nelis ervan overtuigd dat Apps For Ghent een ideale gelegenheid was om hun open strategie te duiden. Luk Verhelst, operaties- en technologiedirecteur van Cultuurnet, werd ook uitgenodigd om te spreken over een API waar voor heel Vlaanderen de opkomende events worden verzameld. Als laatste databron kon de iRail-API gebruikt worden. Deze API kan de historische vertragingen van de drie Gentse stations en de actuele vertrektijden van alle treinen weergeven.

2.3.1 Outerthought

De sponsor van de juryprijs was Outerthought. Outerthought is een bedrijf gevestigd in het technologiepark te Zwijnaarde. Met hun product Lily bouwen ze een NoSQL-framework waarbij data kan worden opgeslagen, gelinkt en geïndexeerd.

Alle statische datasets die beschikbaar waren op Apps For Ghent werden in een Lilycluster gedumpt. Via de Lily-API konden alle gegevens worden opgevraagd.

2.3.2 Xplore+

De sponsor van de publieksprijs was Xplore+. Xplore+ is een bedrijf uit Meldert dat zich focust op business intelligence, semantics en open data. Samen met het IBBT zit Xplore+ ook in het LODGE-project.

2.4 Apps

Er waren 13 groepjes ingeschreven. Elk groepje kreeg de tijd om een presentatie en demo uit te werken. Dit is er uitgekomen:

1. Wakankdoen – SumoCoders – een app die je evenementen aanbeveelt in jouw buurt in de nabije toekomst. `http://wakankdoen.be`

2. VirtualGhent (3DBril) – Johannes – door een bril op te zetten bevind je je plots aan de Nederschelde.

3. Kadee – BitsOfLove – een app die je eerste hulp biedt bij een uitstap met je zoon of dochtertje, opgeplitst in: evenementen, leuke plaatsen en hulp bij ongelukjes.

4. Pendlr – iRail – hoe ver is alles van elkaar verwijderd met het openbaar vervoer? Dit is een visualisatie die haltes van De Lijn in Gent afbeeldt naargelang de werkelijke tijd om er heen te pendelen.

5. Lijnly Planet – Schrödinger's concats – nooit meer eenzaam aan de bushalte. Open de app, en op basis van je locatie wordt er gezocht naar iemand in je buurt. Als iemand naast je ook de app opende, wordt de naam uitgesproken van die persoon.

6. SlimUit – xplore+ – een dag-tot-dag weergave die een mash-up maakt van mogelijke bijkomende informatie: weersvoorspellingen, events...

7. TwinSeats – Joren Six en Luk Verhelst – je wilt graag naar een bepaald concert maar helaas heb je geen vrienden met dezelfde smaak? Via TwinSeats kan je iemand nieuw ontmoeten op het concert.

8. Ik let op mijn lijn – iText – tekenen van buslijnen in PDF.

9. Azumuta – Cedric en Batist – help mee Gent tot betere stad te verheffen: als je iets ziet wat beter kan, neem je een foto en meld je het aan azumuta.

10. Heatmap (OpenCoverage) – Pieter, Bert en Jeroen – een heatmap van GSM-ontvangst in Gent.

11. Stadgids – Jens en Laurens – aan welke halte moet ik uitstappen? Een alarmbel waarschuwt je.

12. Dokter in Gent – Cocoaheads – waar vind ik de dichtsbijzijnde dokter die mij kan helpen in Gent?

13. Gentse Feesten – Creative Geeks – een app voor de Gentse Feesten.

2.5 Feedback en nabespreking

De eerste editie van Apps For Ghent was een voorproef. Een eerste keer een evenement organiseren gaat altijd gepaard met z'n kinderfouten. We vroegen onze genodigden om feedback. Een overzicht waaruit we kunnen leren.

- 13 presentaties van ideeën maakten het op het eind van de dag net iets te langdradig (en dat lag niet aan de onderwerpen)
- Meer data! Er was zeker heel wat data voorzien, maar meer data moet mogelijk zijn
- Op voorhand vermelden welke datasets er worden ontsloten

- Criteria voor de juryprijs vooraf bekendmaken
- Te weinig tijd voor het uitwerken van het idee
- Te weinig kruisbestuiving tussen de teams: sommige teams zijn sterk grafisch, andere op dataverwerkingsvlak
- Te weinig feedback van de dataproviders zelf
- Mensen in 1 zaal zetten zodat er meer samenwerking en interactie mogelijk is

Bij volgende edities van *Apps For X* zal hier rekening mee worden gehouden. In Nederland loop het evenement over een langere periode. Misschien is dit ook hier een goede optie.

2.6 Conclusie

Er valt vast te stellen dat niet alleen data-eigenaars met vragen zitten rond hoe data op een goede manier te ontsluiten vallen, maar dat ook dataconsumenten vragen hebben hoe data om te vormen is naar een model dat kan gebruikt worden in hun toepassing. Datagebruikers op Apps For Ghent begonnen steeds met hetzelfde: het schrijven van een tussenlaag die de ruwe data interpreteert.

We hebben een framework nodig waar een bestaand datasysteem zonder problemen kan worden omgevormd tot een Representational State Transfer (REST)-ful API. Omdat gebruikers zelf als eerste stap een eigen API zullen bouwen, willen we deze al een stap voor zijn. Gebruikers zullen enkel een query moeten opbouwen en die sturen naar een zekere API. Die zal de juiste data van de juiste instanties ophalen en teruggeven.

Om dit te verwezenlijken is er geen “big data”-verwerkingssysteem zoals Lily nodig. Aan de ene kant willen eigenaars een stuk van hun data die intern al in een “big data”-systeem zit ter beschikking stellen aan derden. Op deze data moet feedback kunnen worden gegeven, moet gelinkt kunnen worden aan *The Web Of Things* (zie hoofdstuk 4) en moet een analyse naar gebruik op kunnen worden gedaan.

Aan de andere kant moet dataconsumenten een stap worden uitgespaard. Als data kunnen worden gestructureerd door *The DataTank*, is hun tussenstap overbodig.

Hoofdstuk 3

Het relationele model

De titel van deze scriptie is “een relationele interface voor The DataTank”. Wat betekent “relationeel” nu juist?

3.1 Relaties

Een *relatie* (tabel 3.1) is een datastructuur die bestaat uit een hoofding en een ongeordende reeks van tupels van dezelfde soort.[8] Een *tupel* volgt een zeker voorschrift. De *plaatsigheid* of de *ariteit* (arity in het Engels) is het aantal element dat de hoofding, alsook een tupel, bezit.

Als we spreken over het relationele model hebben we het niet over interlinken van verschillende soorten datasets. We hebben het over een manier waarop data gestructureerd worden.

3.2 Een relationele interface

Als we spreken over een relationele interface denken we uiteraard automatisch aan een Relationeel Databank Management Systeem (RDBMS). Bij een RDBMS kan data worden opgehaald uit een specifieke tabel (of relatie) en kunnen daar operaties op worden uitgevoerd. Bij RDBMS wordt er gebruik gemaakt van een query-taal, meestal SQL.

SQL ondersteunt heel wat basisoperaties voor tabulaire data. De taal heeft haar strepen al verdiend en wordt tot op vandaag in bijna elk softwareprogramma

hoofding	attribuutA	attribuutB	attribuutC
tupel1	waarde1a	waarde1b	waarde1c
tupel2	waarde2a	waarde2b	waarde2c
tupel3	waarde3a	waarde3b	waarde3c

Tabel 3.1: Een ternaire relatie

gebruikt om gegevens op te slaan en op te vragen, zij het SQLite, MySQL, Oracle, PostgreSQL of eender welk ander RDBMS waarvan het gebruik wijd verspreid is van smartphones tot webscale omgevingen.

SQL moet, o.a. uit veiligheidsoverwegingen, worden opgeroepen door scripts en programma's op dezelfde server als de RDBMS. Om een databank ruimer ter beschikking te stellen moeten we gebruik maken van andere technologieën, die vaak geen relationele interface ter beschikking stellen.

3.2.1 REST

REST vereenvoudigt het raadplegen van online data door handig gebruik te maken van het HyperText Transfer Protocol (HTTP). Hierbij wordt een databron gelinkt aan een online Uniform Resource Locator (URL). De manier waarop relationele vragen gesteld kunnen worden verschilt van implementatie tot implementatie. URL's kunnen een collectie of een resource zijn. Een collectie is een verzameling van resources waar data uit kunnen worden gehaald. Heel wat websites en web-services worden vandaag volgens dit principe gebouwd.

Van een relationele interface kan hier niet echt worden gesproken. Toch is het mogelijk om snel data te identificeren en op te vragen zoals in dit voorbeeld:

```
http://example.be/collections/collection1/resource
```

Het mag duidelijk zijn dat deze URI een resource beschrijft die zich in de collectie met naam "collection1", onderdeel van de collectie "collection", bevindt. In hoofdstuk 4.3.1 bespreken we hoe REST hand in hand gaat met *Het Semantische Web* en hoe dat in z'n werk gaat.

3.2.2 HTSQL

Hyper Text Structured Query Language (HTSQL) [9] is net zoals REST een makkelijke manier om data te raadplegen via HTTP. In plaats van elke bron een unieke URI te geven stellen we zelf de logica op, eerder zoals bij SQL, om tot een bepaalde output te komen. In tegenstelling tot REST verzorgt HTSQL enkel tabulaire data opgeslagen in een RDBMS. Het voordeel van HTSQL is dat het wel een relatief complete relationele interface verzorgt.

Bijvoorbeeld:

```
http://example.be/school{name, count(department)}
       ?count(department)>avg(@school.count(department))
```

Hier worden alle schoolnamen opgevraagd, samen met het aantal departementen die een school hebben met die naam. Na het vraagteken vinden we een filter, of in hun terminologie een zeef (sieve), waarbij enkel die scholen mogen weergegeven worden die meer dan het gemiddelde aantal departementen overstijgen.

Zo wordt elke query een op een omgezet tot SQL-query. Het resultaat daarvan wordt weergegeven aan de client-applicatie. Bijvoorbeeld deze JSON-output:

```
{
 meta: [
  {
    title: "name",
    domain: "string"
  },
  {
    title: "count(department)",
    domain: "number"
  }
 ],
 data: [
  [
    "School of Business",
    3
  ],
  [
    "School of Engineering",
    4
  ], ...
 ]
}
```

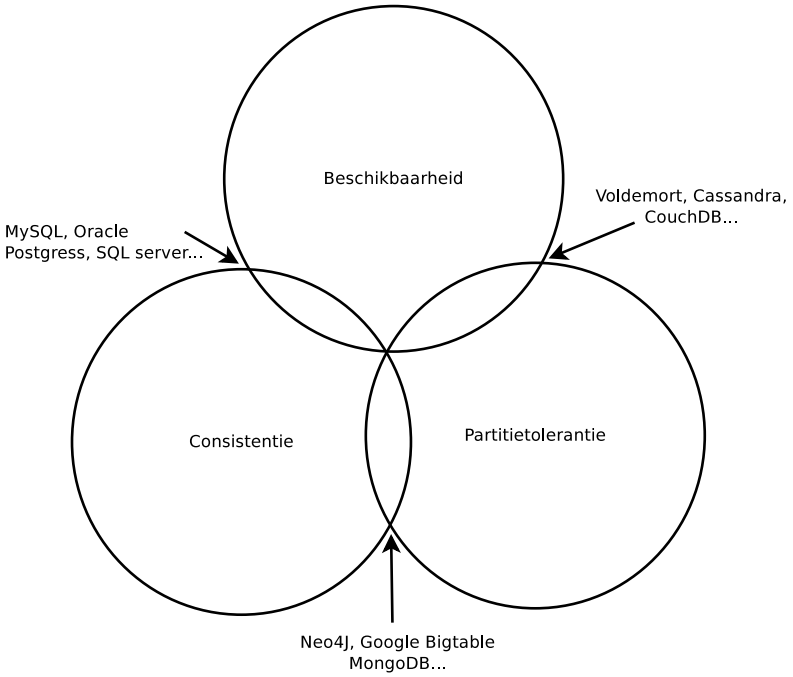
3.3 Relationele databanken in vraag gesteld

Wat is er verkeerd met relationele databanken? Niets, concludeert Eben Hewitt in "Cassandra, a definitive guide" [10], een handboek over het NoSQL-framework. Maar stellen we wel de juiste vraag? Misschien moeten we eerst even het probleem beter specificeren.

3.3.1 CAP ou pas CAP?

Bedrijven volgen vaak hetzelfde stramien: eens de capaciteit van hun server is bereikt, zal er worden geoptimaliseerd. Nadat er op softwarevlak niets meer te optimaliseren valt, zal er verticaal worden geschaald: Random Access Memory (RAM) geheugen uitbreiden, oude harde schijven door snellere vervangen, upgraden naar snellere processoren enzovoort, tot ook hier de limiet bereikt is. Dit leidt ons tot een nieuw probleem: hoe zorgen we voor webscale gedistribueerde systemen?

In de CAP-stelling [11] stelt Brewer dat er bij een databank drie hoofdparameters zijn waar we rekening mee moeten houden: consistentie (*consistency*), tolerantie van partities (*partition tolerance*) en beschikbaarheid (*availability*). De stelling beweert dat slechts twee van de drie vereisten samen voor 100% kunnen voorkomen; wat later formeel werd bevestigd door Seth Gilbert en Nancy Lynch[12]. Ons relationeel model hield tot nu toe perfect rekening met consistentie en beschikbaarheid.



Figuur 3.1: De CAP-stelling [10]

- (C) Consistentie: alle gelijke leesacties op de database zullen dezelfde informatie weergeven.
- (A) Beschikbaarheid: als er een server uitvalt, kan er nog altijd een back-up-server het werk overnemen.
- (P) Partitietolerantie: de databank kan worden gesplitst over verschillende machines. Ze blijft functioneren als er een deel uitvalt.

Nu we partitietolerantie willen toevoegen, kunnen we twee zaken doen. Een eerste aanpak is om dit in ons relationeel model te passen. *Sharding* omvat drie strategieën[1] en is een welgekende methode om dit te verwezenlijken. Het zal echter nooit zorgen voor 100% tolerantie. De tweede aanpak is om, volgend uit de CAP-stelling, ofwel 100% beschikbaarheid ofwel 100% consistentie te laten vallen in ruil voor 100% partitietolerantie.

3.3.2 100% partitietolerantie

Er zijn heel wat projecten die momenteel voor de tweede aanpak hebben gekozen. Om de metafoor te gebruiken waarmee we dit hoofdstuk aanvingen: ze hebben niet gekozen om een sneller paard te implementeren, maar misschien vinden ze wel een auto uit. Aangezien het Web sneller en sneller aan het groeien is[2],

[1] Feature-based shard door Randy Shoup (eBay), Key-based sharding en lookup table. [10]

[2] Ter illustratie: de film Avatar nam 1PB (10^6GB) in beslag en op YouTube worden per dag meer dan 50.000 uren video geüpload (cijfers van het najaar 2010). [10]

is er nood aan meer schaalbare en meer gedistribueerde systemen. Systemen als *Cassandra*, *Voldemort* of *CouchDB* geven de voorkeur aan beschikbaarheid. Consistentie wordt dan een minder schaalbare parameter en zal de oorzaak zijn van veel netwerkverkeer bij hoge load, aangezien er zware synchronisatiescripts zullen moeten draaien.

Systemen die kiezen voor consistentie zijn bijvoorbeeld *Neo4J* en *Google Bigtable*.

3.3.3 Datarepresentatie

Cassandra is vooral een goed antwoord op de vraag "Wat als schaalbaarheid geen probleem is?". Het is gebouwd met de gedachte dat de onderliggende data gedistribueerd beschikbaar en weggeschreven moeten worden. Het laat iemand toe eerst een query op te stellen en er dan de data rond te voorzien, in plaats van eerst de data te modelleren en er achteraf complexe join queries rond te bouwen. Cassandra heeft geen querytaal. In plaats van SQL maakt het gebruik van Remote Procedure Calls (RPC) om data op te halen in de vorm van ijle hashmaps.

Hoofdstuk 4

W3C en het Semantische Web

Het World Wide Web Consortium (W3C) is een vereniging geleid door Tim Berners-Lee, vaak beschouwd als de uitvinder van het Web, die webstandaarden opstelt. Zijn geesteskind is het "Semantische Web".

Semantiek, van het Griekse semantiká, is de studie van de betekenis. Ze wordt in veel vakgebieden, zoals vooral in de linguistiek, aangehaald om complexere problemen op te lossen. Wij zullen het in de rest van de masterproef enkel hebben over de computationele semantiek, en meer specifiek over het Semantische Web.

Goed om in het achterhoofd te houden bij de rest van deze uitleg is dat semantiek geen doel op zich is. Het semantisch annoteren van data is een tussenstap die handig kan zijn voor sommige toepassingen. Dit wordt vaak uit het oog verloren.

4.1 Semantiek en the Web Of Things

Het Semantische Web, ook wel benoemd als Web 3.0 en The Web Of Things, wordt vaak cryptisch omschreven.

Volgens semanticweb.org is het: "The Semantic Web is the extension of the World Wide Web that enables people to share content beyond the boundaries of applications and websites."

Volgens de Engelstalige Wikipedia is het: "The Semantic Web is a man-made woven web of data that facilitates machines to understand the semantics, or meaning, of information on the World Wide Web."

Laten we deze beide definities even bezinken. Het Semantische Web is een uitbreiding op het oorspronkelijke World Wide Web en wordt nu ook wel het Web Of Things genoemd. Vroeger werden termen als *hypertext* aangehaald om aan te duiden wat het Web 1.0 nu eigenlijk was. Via hyperlinks werd een web geweven van computers die informatie met elkaar kunnen delen. Het schoolvoorbeeld dat steeds werd aangehaald was dat van een receptenpagina. Alice kon dan van op haar pagina naar Bobs receptenpagina linken.

Het Semantische Web wordt ook het Web Of Things genoemd omdat nu een URI geen verwijzing meer is naar een bron van informatie, maar naar wat wij in het vervolg van de scriptie een real-world object zullen noemen. Een real-world object is eender welk ding: een persoon, een evenement, een vervoersmiddel, een gebruiksvoorwerp...

Een ontologie is een beschrijving van een soort ding (zie het als een klassebeschrijving van mogelijke objecten in ObjectGeörienteerd Programmeren (OGP)). Het beschrijft wat er in een representatie van een bepaald object kan zitten.

Een optreden op de Gentse Feesten is een evenement. Een online *ontologie* valt snel te vinden door het Web te doorzoeken naar "event ontology". Met behulp van een zoekmachine komen we op de website `http://linkedevents.org/ontology`. We kunnen over dit optreden zeggen dat het een `http://linkedevents.org/ontology/Event` is. Maar dit optreden is nog meer: het is gelinkt aan de Gentse Feesten. We kunnen dus ook ergens een link leggen met `http://gentsefeesten.be`.

Het Web wordt aan elkaar geweven tot een Web Of Things door relaties te leggen tussen verschillende URI's. Zo kan een computer zelf gaan redeneren (to *reason*) naar wat er mogelijk in een bepaald *ding* zal zitten. Voor academici is dit een uiterst interessant onderzoeksdomein, gezien de mogelijkheden van deze nieuwe *laag* om data te linken. Neurale netwerken kunnen gebouwd worden met een meer toegankelijke gigantische bron van informatie.

Een kleine zijsprong We keren even terug naar onze oorspronkelijke definitie van semantiek: "Semantiek is de studie van de *betekenis*". We kunnen een zin in verschillende talen zeggen, we kunnen ze anders formuleren, we kunnen ironie of sarcasme gebruiken, enzovoort. Zolang de zin dezelfde betekenis behoudt, zijn ze semantisch gelijk. Bij computationele semantiek kan men hetzelfde vaststellen. Een ding kan wel een geolocatie voorstellen, maar welke soort coördinaten er in zitten kan sterk verschillen van dataset tot dataset[1].

Een voorbeeld uit de linguistiek

- De Gentse Feesten zijn een leuk initiatief.
- De 10-daagse feesten gehouden in Gent tijdens juli zijn een leuk initiatief.
- De 10-daagse feesten gehouden in de hoofdstad van Oost-Vlaanderen zijn met zekerheid het omgekeerde van een saai initiatief.

Deze drie zinnen betekenen telkens identiek hetzelfde en er is telkens evenveel informatie uit te halen. De schrijver geeft een eigenschap aan de Gentse Feesten. Noam Chomsky bijvoorbeeld ontwikkelde in de jaren 60 de generatieve grammatica [13] op basis van deze vaststelling: een zin met een bepaalde semantiek (of betekenis) kan naar een andere zin *getransformeerd* worden. Met voldoende transformaties kunnen zaken naar een basisvorm gereduceerd worden waardoor ze vergeleken kunnen worden.

[1] Een oplossing voor dit probleem is standaardisatie. Voor dit specifieke voorbeeld zijn er al verschillende standaarden bekend: WGS84, Lambert72, NAD83 ... Standaardisatie wordt in deze scriptie niet behandeld.

onderwerp	predicaat	object
Een vis	is een	dier
Nemo	is een	vis
http://gentsefeesten.be	is a	http://linkedevents.org /ontology/Event

Tabel 4.1: Voorbeelden van triples

Semantische netten Als we alle puzzelstukken over het hele internet samenleggen, hebben we een groot *semantisch net*. Een semantisch net is een verzameling van beschrijvingen (ontologieën) en objecten waarbij alles op een bepaalde manier gelinkt is. Zo moet het in de toekomst mogelijk worden om elk object te linken met een gelijkaardig object.

4.2 Mark-up- en querytalen

Om een semantisch net over het hele internet te bouwen hebben we nood aan standaarden. Er zijn verschillende door het W3C aanvaarde standaarden in omloop die daartoe bijdragen. We geven een beknopt overzicht.

4.2.1 Semantische formaten

RDF

Het Resource Description Framework (RDF) is de standaard van het W3C om gegevens uit een semantisch net voor te stellen en uit te wisselen. Het maakt gebruik van triples. Triples zijn uitspraken van het subject-predicaat-object formaat. In tabel 4.1 staan enkele voorbeelden.

Opslaan van data Om RDF-data op te slaan maken we gebruik van triplestores. Een triplestore is een stuk software dat gespecialiseerd is in het wegschrijven en opvragen van triples. In zijn eenvoudigste vorm kan een triplestore beschouwd worder als een ternaire relatie (of een tabel met 3 kolommen) met als attributen (of kolomnamen) een subject, een predicaat en een object.

Weergeven van data Om RDF-data weer te geven zijn er verschillende syntaxis. De meest voorkomende is om RDF in XML weer te geven, maar ook best populair zijn talen als Turtle, N3, semantic-JSON, Trix, RDFa...

Alternatieven Hoe simpel de standaard ook, vaak zijn er alternatieve modellen die even goed werken. Een mooi voorbeeld hiervan zijn Topic Maps. Topic Maps, dat gestandardiseerd werd door International Organization for Standardization (ISO) [14], geeft informatie weer aan de hand van onderwerpen, associaties en gebeurtenissen. Een Topic Map kan het beste vergeleken worden met een mindmap. Het grootste verschil met RDF is dat Topic Maps niet alleen

gebruik maakt van triples, maar ook van meerplaatsigheid[2]. Dit leidt ertoe dat, in tegenstelling tot RDF, dat kan worden voorgesteld door een enkelvoudig gerichte graaf, een Topic Map voorgesteld kan worden als een hypergraaf.

Aangezien Topic Maps in het kader van het Semantische Web weinig voorkomen, beperken we ons in de rest van de masterproef tot RDF.

HTML, RDFa en microformats

HTML Websites worden gebouwd aan de hand van een mark-up-taal Hyper Text Markup Language (HTML) genaamd. Deze taal werd al gebruikt toen de eerste browser werd geschreven. Aan de hand van (XML)-achtige tags werd een pagina uitgewisseld over het HTTP-protocol, om op het toestel van de gebruiker een overzichtelijk beeld te geven van de inhoud. Het server- en het client-toestel hebben hier geen idee welke soort informatie ze aanbieden of tonen.

```
<title>Hypertext Links</title>
<h1>Links and Anchors</h1>
A link is the connection between one piece of
<a href=WhatIs.html>hypertext</a> and another.
```

Codefragment 4.1: De broncode van de langst niet aangepaste webpagina op het Web: `http://www.w3.org/History/19921103-hypertext/hypertext/WWW/Link.html` - Tim Berners-Lee publiceerde deze op dinsdag 13 november 1990 15:17:00 GMT

HTML5 Al snel verschenen nieuwe versies van HTML en werden deze op semantisch vlak rijker. De huidige "working draft" van het W3C is HTML5 [15]. Hier is niet alleen de titel van de pagina duidelijker, ook is heel snel de essentie van de pagina begrijpbaar. Als we enkel het artikel zelf willen filteren, de eerste sectie, de volgende pagina, etc. is dit zonder meer mogelijk.

```
<!DOCTYPE html>
<html lang=nl>
  <head>
    <meta charset="utf-8">
    <title>Een HTML5 voorbeeld</title>
  </head>
  <body>
    <header>
      <h1>Een HTML5 voorbeeld</h1>
    </header>
    <nav>
      <a href="pagina1">1</a>
      <a href="pagina2">2</a>
    </nav>
    <header>
      <h2>Ondertitel</h2>
```

[2]Zie ook hoofdstuk 3 over het relationeel model. Een plaatsigheid of ariteit is het aantal attributen in een relatie.

```
    </header>
    <article>
      <section>...<section>
    </article>
    <aside>...</aside>
    <footer>&copy; 2011</footer>
  </body>
</html>
```

Codefragment 4.2: Een voorbeeld van een HTML5-pagina

In codefragment 4.2 staat een eenvoudig voorbeeld. De header is duidelijk herkenbaar, de navigatie kan zonder problemen geëxtraheerd worden en ook de stukken van een artikel zijn onmiddellijk abstraheerbaar. Dat dit een enorm groot voordeel betekent voor zoekmachines, die de essentie uit een pagina willen halen, is duidelijk.

Bij standaard HTML5 kunnen elementen niet gelinkt worden aan een externe ontologie. Wel omarmt het de idee om meer betekenis te geven aan een pagina door deze semantisch te annoteren als deel van de mark-up-taal zelf. HTML5 is goed om pagina's inhoud te geven, maar het schiet tekort om data zelf te annoteren. Om meer betekenis te geven aan bepaalde inhouden op webpagina's zijn er andere middelen voor handen:

RDFa Om webpagina's te verrijken met semantische data kan gekozen worden voor Resource Description Framework in attributes (RDFa). RDFa is helaas enkel compatibel met XHTML omdat het XML-namespaces ondersteunt.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML+RDFa 1.0//EN"
  "http://www.w3.org/MarkUp/DTD/xhtml-rdfa-1.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:foaf="http://xmlns.com/foaf/0.1/"
      xmlns:dc="http://purl.org/dc/elements/1.1/"
      version="XHTML+RDFa 1.0" xml:lang="en">
  <head>
    <title>John's Home Page</title>
    <base href="http://example.org/john-d/" />
    <meta property="dc:creator" content="Jonathan Doe" />
    <link rel="foaf:primaryTopic"
          href="http://example.org/john-d/#me" />
  </head>
  <body about="http://example.org/john-d/#me">
    <h1>John's Home Page</h1>
    <p>My name is <span property="foaf:nick">John D</span>
       and I like <a href="http://www.neubauten.org/"
                     rel="foaf:interest"
                     xml:lang="de">Einsturzende Neubauten
                  </a>.
    </p>
    <p>
```

```
    My <span rel="foaf:interest"
            resource="urn:ISBN:0752820907">favorite
    book is the inspiring <span about="urn:ISBN:0752820907">
    <cite property="dc:title">Weaving the Web</cite> by
    <span property="dc:creator">Tim Berners-Lee</span></span>
   </p>
  </body>
</html>
```

Codefragment 4.3: Een voorbeeld van een RDFa website die de DERI ontologieën gebruikt.

In codefragment 4.3 wordt het concept duidelijk. In de *DOCTYPE* wordt duidelijk dat we te maken hebben met een RDFa pagina. In de HTML-tag definiëren we de XML-namespaces waar we gebruik van zullen maken. Dit zijn tevens de afkortingen die we gebruiken in onze tripels. In de *body* van de pagina wordt op verschillende manieren informatie gelinkt aan het Semantische Web: in een *meta*-tag, in een property attribuut van een *span*- of *cite*-tag, enzovoort.

Het probleem met RDFa is dat elke website eerst moet omgezet worden naar XHTML als dit nog niet werd gedaan. Dit is voor veel websites een te hoge drempel om hierop over te schakelen.

Open Graph Protocol Facebook ontwikkelde onafhankelijk van het W3C ook zijn eigen systeem om het Web interlinkbaar te maken: het *Open Graph Protocol*. Aangezien bleek dat gelijkaardige design principes werden gebruikt, besliste Facebook om zijn Open Graph specificatie aan te passen aan de RDF standaard. Ze verschillen van RDFa op het vlak dat ze niet verwachten dat een pagina aan de XHTML-syntax voldoet [16]. Het Open Graph Protocol dient echter enkel om meta-data aan pagina's toe te voegen. Voor deze metadata moet hun ontologie worden gebruikt, gespecificeerd op `http://ogp.me/ns#`.

```
<html prefix="og: http://ogp.me/ns#">
 <head>
   <title>...</title>
   <meta property="og:audio"
        content="http://example.com/bond/theme.mp3" />
   <meta property="og:description"
        content="Sean Connery found fame and fortune as the
        suave, sophisticated British agent, James Bond." />
 </head>
 <body>
   ...
 </body>
</html>
```

Codefragment 4.4: Een voorbeeld van open graph meta tags

In codefragment 4.4 wordt duidelijk waarom het Open Graph Protocol niet vereist dat XHTML wordt gebruikt: er worden enkel *meta*-tags gebruikt om de *hele* pagina te annoteren. De prefix van de ontologie wordt toegevoegd in de *html*-tag.

Microformats Eerder werd al besproken dat pagina's en hun inhoud volledig kunnen geannoteerd worden mits er gebruik gemaakt wordt van XHTML (RDFa). Pagina's kunnen ook zonder XHTML geannoteerd worden d.m.v. het Open Graph Protocol van Facebook, maar dit draait enkel rond de pagina zelf. Ook de structuur zelf semantisch vastleggen kan d.m.v. HTML5.

Aparte elementen binnenin de inhoud van een pagina annoteren zonder XHTML is nog steeds een vraagteken. Om dit op te lossen en zaken zoals een evenement of een persoon aan te duiden binnen een bepaalde webpagina zijn *microformats* in het leven geroepen. Met een beperkte aanpassing aan een HTML-pagina kan er toch betekenis aan worden toegevoegd.

```
<div id="hcard-Kris" class="vcard">
    <span class="fn">Kris</span>
    <div class="org">Hogent</div>
    <div class="adr">
        <span class="locality">Gent</span>,
        <span class="postal-code">9000</span>
        <span class="country-name">Belgium</span>
    </div>
</div>
```

Codefragment 4.5: Een voorbeeld van microformats. Meerbepaald een hcard.

In codefragment 4.5 wordt de eenvoud van een microformat duidelijk. Een visitekaartje bijvoorbeeld kan in eender welke HTML-pagina worden opgenomen. De mapping aan een ontologie en de specificatie worden besproken op `http://microformats.org`.

Andere initiatieven Er zijn nog tal van andere initiatieven om data semantisch te annoteren. Zo wordt *Atom* bijvoorbeeld gebruikt om nieuws-sites semantisch te verrijken.

De voornaamste initiatieven om HTML semantisch uit te breiden, zowel van meta-data als van de achterliggende data, zijn besproken. Het volgende hoofdstuk zal de pure voorstelling van semantische data behandelen.

Scheiden van de data- en presentatielaag

Het Web is tot nu toe enkel bekeken als een web van gehyperlinkte pagina's. Er bestaan echter nog andere toepassingen over het HTTP-protocol die een abstractie gaan maken tussen een datalaag en een presentatielaag. Een van de eerste initiatieven was XML als datalaag en eXtensible Stylesheet Language Transformations (XSLT) als presentatielaag. Surfen naar een website zou dan surfen naar een XML bestand worden met de pure data er in, via XSLT zou dan de XML worden getransformeerd tot een HTML-pagina. Tegelijkertijd was deze data zonder problemen direct toegankelijk voor desktopapplicaties, mobiele apps, of andere webtoepassingen.

In een tijd waar mobiele devices de kop opsteken hebben we meer en meer de nood om dezelfde data voor te stellen op verschillende platformen. Een abstractielaag is geen overbodige luxe meer. In plaats dat URLs louter HTML

weergeven, gaan URL's nu ook pure data weergeven. Deze data kan weergegeven worden in verschillende formaten: JSON, XML, CSV... en binnen deze formaten bestaat er nog verschillende syntaxis. Om deze zogeheten web-services ook semantisch verrijkte datasets te laten teruggeven, bestaan er nog andere formaten.

RDF/XML De meest voorkomende vorm om RDF voor te stellen is een XML-formaat waarbij uitgebreid gebruik wordt gemaakt van namespaces.

```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:wgs84="http://www.w3.org/2003/01/geo/wgs84_pos#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
   xmlns:schema="http://schema.org/">
  <rdf:Description rdf:about="http://.../vienna/libraries/13122">
    <schema:Library rdf:about="http://.../vienna/libraries/13122">
    <wgs84:lat rdf:datatype="xsd:decimal">
       48.174127098714
    </wgs84:lat>
    <wgs84:long rdf:datatype="xsd:decimal">
       16.411856553663
    </wgs84:long>
  </rdf:Description>
</rdf:RDF>
```

Codefragment 4.6: Een voorbeeld van RDF/XML

In codefragment 4.6 wordt een bibliotheek van de stad Wenen weergegeven. In het root-element vinden we de namespaces terug waar we in het document gebruik van zullen maken. Vervolgens wordt een beschrijving gegeven van een bepaalde URI. Bibliotheek met id 13122 heeft een longitude en een latitude.

Als de lezer dit wil testen, kan er gebruik gemaakt worden van de demo-instantie op `http://datatank.demo.ibbt.be`.

Andere standaarden Er bestaan ook andere niet-XML-standaarden om RDF weer te geven zoals Turtle, N3, N-Triples, RJson, enzovoort. Telkens ondersteunen ze ook namespaces om op een eenvoudige manier tripels weer te geven.

4.2.2 SPARQL

SPARQL Protocol and RDF Query Language (SPARQL) [17] is een protocol en querytaal die bovenop een triplestore toelaat om vragen te stellen. Ze is ontworpen om op web-scale data te doorzoeken. SPARQL werd in 2008 gestandaardiseerd door het W3C [18].

Een triplestore is een databank die geoptimaliseerd is om semantische tripels te bewaren en op te vragen. Op quasi elke triplestore kan een SPARQL-endpoint geplaatst worden. Frameworks die een SPARQL-endpoint voorzien zijn bijvoorbeeld Virtuoso, 4-Store, Sesame ...

```
select distinct *
where {
    ?bibliotheek a schema:Library .
    ?bibliotheek geo:lat ?lat .
    ?bibliotheek geo:long ?long .
}
```

Codefragment 4.7: Een SPARQL query om alle bibliotheken in Wenen op te halen en weer te geven.

In codefragment 4.7 staat een heel eenvoudig voorbeeldje van een SPARQL-query. Normaal worden eerst de namespaces zoals *geo* en *schema* gedefinieerd net zoals dat bij semantische formaten wordt gedaan. Vervolgens worden alle elementen geselecteerd (*) waarvoor geldt dat het element in kwestie een bibliotheek is. Een variabele wordt aangeduid met een vraagteken.

Deze code zal bij uitvoering een output geven in een semantisch formaat. Codefragment 4.6 komt dicht bij de werkelijke output van deze query.

Een huidige beperking van SPARQL is dat het voorlopig nog enkel toelaat om queries uit te voeren op een triplestore. Er kunnen dus geen queries worden gedaan waarbij data van buitenaf geaggregeerd worden met de data in de lokale triplestore.

4.3 Semantiek, REST en HTTP Range 14 issue

Er wordt een kort overzicht van het REST-model[3] gegeven. Vervolgens wordt de mogelijke samenwerking met het Semantische Web en het beruchte HTTP Range 14 probleem besproken.

4.3.1 REST

REST is, net zoals Simple Object Access Protocol (SOAP), een beschrijving van een software-model om data uit te wisselen over het HTTP 1.0 en HTTP 1.1 protocol (RFC 2616 [19]). Een webservice die voldoet aan de REST-voorwaarden wordt RESTful genoemd.

Staatloos Net zoals HTTP is ook REST volledig staatloos. Dit wil zeggen dat er tussen twee oproepen verzonden door hetzelfde toestel geen relatie zal kunnen worden gelegd door de server. De populariteit van REST is vooral te danken aan het feit dat dit software schaalbaar maakt: er moet immers niets in het werkgeheugen blijven hangen. Als de load op de server verhoogt, moet de software die zorgt voor de REST-functionaliteit niet geüpdatet worden.

[3]Aangezien dit model al in het masterjaar van deze opleiding grondig werd besproken, behandelen we slechts de aspecten die relevant zijn voor The DataTank.

URI	GET	PUT	POST	DELETE
Bron	Haal een voorstelling van de bron op.	Voer deze bron in of wijzig de huidige bron.	Behandel de bron als een verzameling bronnen van zijn type en voeg een nieuwe bron toe.	Verwijder deze bron uit de verzameling.
Verzameling	Haal een lijst op van de bronnen binnen deze verzameling.	Vervang de hele verzameling door een nieuwe verzameling bronnen.	Voeg een nieuwe bron toe aan de verzameling.	Verwijder de hele verzameling.

Tabel 4.2: HTTP methodes en hun functie binnen REST. [20]

Unieke URL Afgezien van de technische voordelen voor systeembeheerders, biedt REST ook heel wat high-level richtlijnen die handig van pas kunnen komen. Elke "resource" of databron wordt geïdentificeerd met een URI. Ernaar surfen levert een output op die de dataset bevat. Dit kan JSON zijn of XML of Yet Another MarkUp Language (YAML), maar evengoed HTML.

Collectie vs. bron RESTful webservices doen hun best om zo gebruiksvriendelijk mogelijke URLs te hanteren. Hiervoor maakt REST een onderscheid tussen de databronnen of "resources" zelf en collecties van databronnen. Surfen naar de URI van een collectie levert een oplijsting op van de databronnen die deze collectie bezit.

Vier functies Zoals eerder vermeld maakt REST gebruik van de functies van HTTP. HTTP biedt voldoende methodes om een read-write web-service aan te bieden. REST maakt hier handig gebruik van slechts vier functies: POST, GET, DELETE, PUT. Deze verzorgen samen een Create Read Update Delete (CRUD) interface zoals in tabel 4.2.

4.3.2 Het Semantische Web en REST

REST en het Semantische Web zijn twee zaken die elkaar kunnen aanvullen. Het Semantische Web heeft namelijk URI's nodig om dingen te identificeren. Deze URI's opbouwen volgens REST is een voor de hand liggende keuze.

4.3.3 HTTP Range 14

Als we dingen online aan elkaar linken, hebben we verschillende soorten URI's nodig. Enerzijds moeten real-world objects een identificatie krijgen. Anderzijds moeten hun representaties een andere URL krijgen.

Bij het Web Of Things zullen we een URI geven aan real-world objects. Een trein krijgt zo bijvoorbeeld http://nmbs.be/train/1 als identificatie. Deze URI hoeft niet als URL te bestaan. Dit is enkel een identificator om later ook andere objecten hiernaar te linken. Een mogelijke representatie van deze trein zal de doorkomsttijden in enkele stations weergeven. Een andere representatie zal het oliepeil weergeven.

Als data worden ontsloten, moet er voor elk ding een nieuwe URI komen waar representaties aan gelinkt zijn. Hoe deze URL's en URI's worden opgebouwd wordt binnen het W3C het HTTP Range 14 issue genoemd. Tim Berners-Lee stelt zelf voor om je neus te volgen hoe dit concreet wordt gedaan. [21] Voor The DataTank gaan we daar in deel II verder op in.

4.4 Het Semantische Web vandaag en The DataTank

Het Semantische Web staat in 2012 nog in zijn kinderschoenen. Websites worden nog niet de facto voorzien van RDFa en ook in The DataTank zijn bij het toevoegen van een nieuwe resource de data nog niet gelinkt aan een ontologie. Toch is er een grote opmars aan de gang: Facebook lanceerde het Open Graph Protocol (RDFa), Google zoekt op semantische sitemaps; in Drupal wordt vanaf versie 7 een basis taxonomie meegeleverd en ook bij de software achter Wikipedia (Mediawiki) is er een grote opmars aan plug-ins om semantiek toe te voegen. Het huidige Semantische Web wordt regelmatig in kaart gebracht. De toestand van het Semantische Web in september 2011 wordt voorgesteld in fig. 4.1.

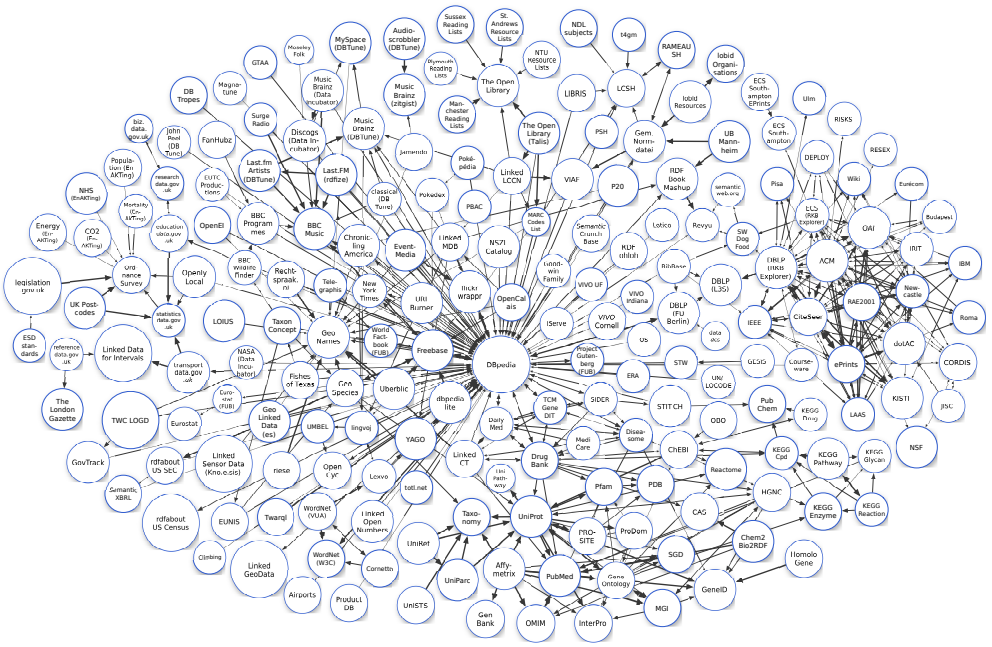
Semantiek inbrengen op een dataset is een uiterst zware taak. Daarom zal The DataTank de brug bouwen tussen de 3e tot de 5e ster van open data. Data-eigenaars zullen niet belast worden met het linken van hun dataset aan een ontologie, maar de data zal verrijkt worden door de gebruikers zelf.

Op Open Government Data Camp (ogdcamp) in Warschau en op het DataSalon in de Vooruit te Gent presenteerden Miel Vander Sande en ik Follow The Stars[1]. The DataTank zorgt ervoor dat de barrière om naar een volgende ster te gaan drastisch wordt verlaagd door crowd sourcing. We leren van onze gebruikers doordat ze aan de hand van een Graphical User Interface (GUI) resources kunnen linken aan andere URI's.

```
"datum","start","stop","uitleg","locatie"
"18/2/2011","12:00:00","18:00:00","G...","G...D"
"19/2/2011","12:00:00","18:00:00","U...","H...e"
"21/2/2011","11:00:00","11:00:00","M...","-"
"26/2/2011","16:00:00","22:00:00","E...","H...e"
"5/3/2011","11:00:00","17:00:00","I...","H...e"
...
```

Codefragment 4.8: Deze CSV-file wordt ingeladen op http://pieter.demo.thedatatank.com/thesis/diary

Een van de belangrijkste zaken van Linked Open Data (LOD) is dat we elke dataset kunnen aanspreken door middel van URI's. Met The DataTank wordt dit



Figuur 4.1: The Linked Open Data cloud in September 2011 (© w3c - Creative Commons By Sa: attribute and share alike)

onmiddellijk mogelijk. Als bijvoorbeeld een Comma-Separated Values (CSV)-file aan een The DataTank-instantie wordt toegevoegd zonder primary key, worden URI's aangemaakt voor ieder stuk binnen deze dataset. real-world object en representatie worden duidelijk van elkaar gescheiden (HTTP Range 14 issue) door middel van de format specificator (.json, .about, .xml).

```
[
  {
    "datum":"18\/2\/2011",
    "start":"12:00:00",
    "stop":"18:00:00",
    "uitleg":"Git opgezet en aanmaken documenten.",
    "locatie":"Gent - ByB D"
  }
]
```

Codefragment 4.9: Deze resource staat live op http://pieter.demo.thedatatank.com/thesis/diary/0.json

Content negotiation Als `.about` werd gekozen, waar overigens direct wordt naar verwezen vanaf een real-world object URI, zal The DataTank automatisch een formatter kiezen op basis van de HTTP-accept-header. Deze header wordt door client-applicaties, zoals een browser, meegegeven om een voorkeur te geven aan een bepaald formaat.

De Accept-header ziet er als volgt uit bij Mozilla Firefox: “Accept: text/html, application/xhtml+xml, application/xml;q=0.9, */*;q=0.8”. Dit wil zeggen dat Firefox de voorkeur geeft dan een HTML-bestand of een XHTML-bestand. Indien niet beschikbaar mag de server ook XML weergeven. In het slechtste geval mag de server de client eender wat terug sturen. Als er dus niks na de “;” werd meegegeven, krijgt dit automatisch een gewicht van 1.0. Hoe lager de q, hoe lager de prioriteit.

Aangezien SPARQL nog niet toelaat om verschillende online databronnen te combineren, maken we in het verdere verloop van de masterproef geen gebruik van SPARQL. Toch kan de gebruiker zelf snel een SPARQL-endpoint opzetten door de RDF-data die kan worden opgevraagd via de REST-interface in een triplestore op te slaan. Uiteraard moet dan regelmatig een nieuwe versie worden opgehaald naargelang de data in The DataTank wijzigt.

Een tweede reden om nog niet voor SPARQL te kiezen is dat een dataset in The DataTank niet automatisch een ontologie krijgt toegewezen. Basis-acties zoals sorteren, selecteren, filteren, enzovoort kunnen overigens beter worden uitgevoerd door een (niet-semantische) interface die wel al goed ingeburgerd is bij dataconsumenten. Een REST-achtige interface aanbieden met relationele mogelijkheden kan voor een dataconsument de nood om een eigen web-service op te zetten tenietdoen. Deze relationele interface wordt besproken in hoofdstuk 7.

Deel II

Analyse en implementatie

Hoofdstuk 5

The DataTank

“

The adverse impact on development productivity of requiring programmers to navigate along access paths to reach target data [...] was enormous. In addition, it was not possible to make slight changes in the layout in storage without simultaneously having to revise all programs that relied on the previous structure. [...] As a result, far too much manpower was being invested in continual (and avoidable) maintenance of application programs.

E.F. Codd [8],
de bezieler van het relationeel model waarop
relationele databanken steunen (1970)

”

Bij aanvang van de masterproef in februari 2011 was er van The DataTank in zijn huidige gedaante nog geen sprake. De bedoeling van deze masterproef was om gedurende de hele masterproefperiode mee te bouwen aan The DataTank. Er moest worden toegezien op de architectuur om ervoor te zorgen dat een relationele interface implementeren zo eenvoudig mogelijk zou worden. In november werd uiteindelijk ook die relationele interface verzorgd.

De architectuur van The DataTank, geïmplementeerd in PHP5.3, werd gebouwd naar de verschillende problemen die hij oplost. De problemen richten zich telkens concreet op open data (zie hoofdstuk 1).

5.1 Controle op data

Bij het ontsluiten van data stellen er zich problemen bij de consistentie. Als een dataset integraal wordt doorgegeven aan een derde partij moet de data-eigenaar, telkens een update wordt doorgevoerd, ervoor zorgen dat iedere derde

op de hoogte wordt gesteld. Aangezien alle gebruikers hier gekend zouden moeten zijn, zal waarschijnlijk een gebruikerscontract opgesteld worden die updates garandeert. Dit is dan weer een probleem voor open data, aangezien deze data niet meer automatisch opgevraagd worden door een Semantic Web-query (zie sectie 4.2.2).

Tenzij een dataset historische data zoals statistieken bevat, is dit een zeer groot euvel dat veel bedrijven en organisaties weerhoudt om de stap naar een opendatabeleid te zetten. The DataTank lost dit probleem op door nooit data op te slaan, hoogstens even te cachen, maar door data rechtstreeks te halen uit de initiële bron zelf. Dit zorgt ervoor dat alle data die geleverd worden aan derde partijen direct kunnen worden geüpdatet bij iedere client-applicatie.

Caching, of het tijdelijk bijhouden van eerder opgezochte informatie, is een techniek om data sneller toegankelijk te maken binnen een softwareprogramma. Eenmaal iets is opgevraagd wordt dit tijdelijk opgeslagen in het geheugen. Als dit de volgende keer wordt opgevraagd halen we de waarde die opgeslagen werd binnen een zekere tijdspanne. Dit zorgt voor een opmerkelijke tijdsverbetering en load-vermindering bij data die regelmatig kort na elkaar geraadpleegd wordt. Bij The DataTank wordt caching, hoewel het makkelijk is een ander systeem te implementeren, verwezenlijkt door memcached. [22]

Om de controle op data te behouden moet bij real-time en dynamische data ervoor gezorgd worden dat de vervaltijd van een gecachete sleutel kleiner is dan de tijd waarin de externe data verandert. De vertraginsinformatie op de server van de Belgische spoorwegen worden om de minuut bijgewerkt. De vervaltijd op data.iRail.be is dan ook een minuut. Bij statische data kan de cache-tijd onbeperkt zijn. Dit wil niet zeggen dat die informatie altijd in de cache blijft staan. Als ze niet recent werd opgevraagd wordt ze sowieso verwijderd als er plaats nodig is.

Caching bij The DataTank gebeurt door een singleton “Cache”. Deze singleton heeft 3 functies: get, set en delete. Welk cachingsysteem deze klasse moet gebruiken ligt vast in de configuratie. Momenteel ondersteund zijn: NoCache en MemCached. Een nieuwe implementatie maken is dus eenvoudig.

```
$c = Cache::getInstance();
$element = $c->get($id);
if(is_null($element)){
    $element = get_the_right_data();
    $c->set($id,$element,$timeout);
}
```

Codefragment 5.1: Een voorbeeld van hoe de cache kan worden gebruikt.

In codefragment 5.1 wordt in een codefragment in PHP getoond hoe de cache kan worden aangesproken eender waar in het project. Nadat een instantiatie werd opgevraagd kan het element opgehaald worden. Is dit element leeg, dan kan de data op de normale manier worden opgehaald. Daarna wordt de data in de cache opgeslagen voor een nader te bepalen tijdspanne.

5.2 Geen dubbel werk: webformaten

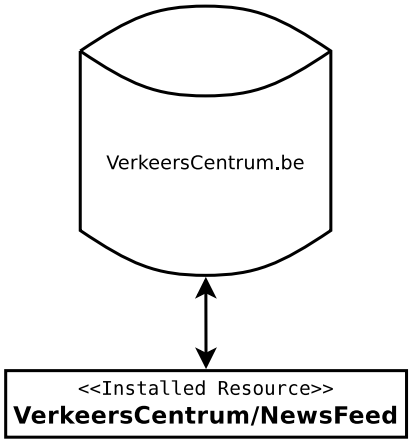
Als data moeten worden verwerkt door dataconsumenten, worden die in allerlei verschillende formaten aangeleverd: XLS, CSV, Shape, KML, enzovoort. Aangezien ontwikkelaars graag data in een eenzijdig formaat krijgen, zijn formaten als XML of JSON in het leven geroepen, ook wel webformaten genoemd.

Volgens REST worden datasets die beschikbaar worden gesteld via URL's ook *resources* genoemd. In The DataTank zal deze naam nog vaak terugkomen. Om datasets in The DataTank binnen te halen hebben we drie verschillende mogelijkheden:

1. Installed resources (fig. 5.1),
2. Generic resources (fig. 5.2) en
3. Remote resources (fig. 5.3).

5.2.1 Installed resources

Geïnstalleerde resources (fig. 5.1) zijn stukken code die data, typisch niet optimaal gestructureerd, van een bepaalde locatie zal afhalen en in een bepaalde structuur in het werkgeheugen zal opslaan. De structuur hangt volledig af van wat de ontwikkelaar beslist.



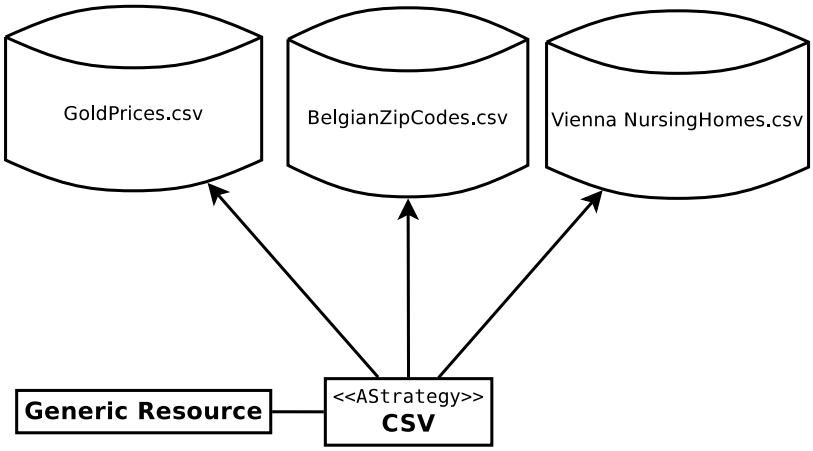
Figuur 5.1: Een voorbeeld van een geïnstalleerde resource in UML: VerkeersCentrum

Typisch worden installed resources gebruikt als webscraper. Ze kunnen eenmaal worden geschreven om een specifieke bron aan te spreken.

5.2.2 Generic resources

Generic resources (fig. 5.2) hangen steeds af van een strategie om een externe HTTP inhoud uit te lezen. Net zoals bij installed resources zal een programmeur

een abstracte klasse implementeren om data binnen te halen. Nu echter gaat het om relatief gestructureerde formaten waar heel wat datasets voor bestaan.



Figuur 5.2: Een voorbeeld van een generieke resource

Een veelgebruikte generic resource strategy is bijvoorbeeld CSV. Een nieuwe CSV-resource kan worden toegevoegd door een PUT-request te doen naar een nog onbestaande URL binnen het domein. Als parameters worden de URL en de soort resource meegegeven.

5.2.3 Remote resources

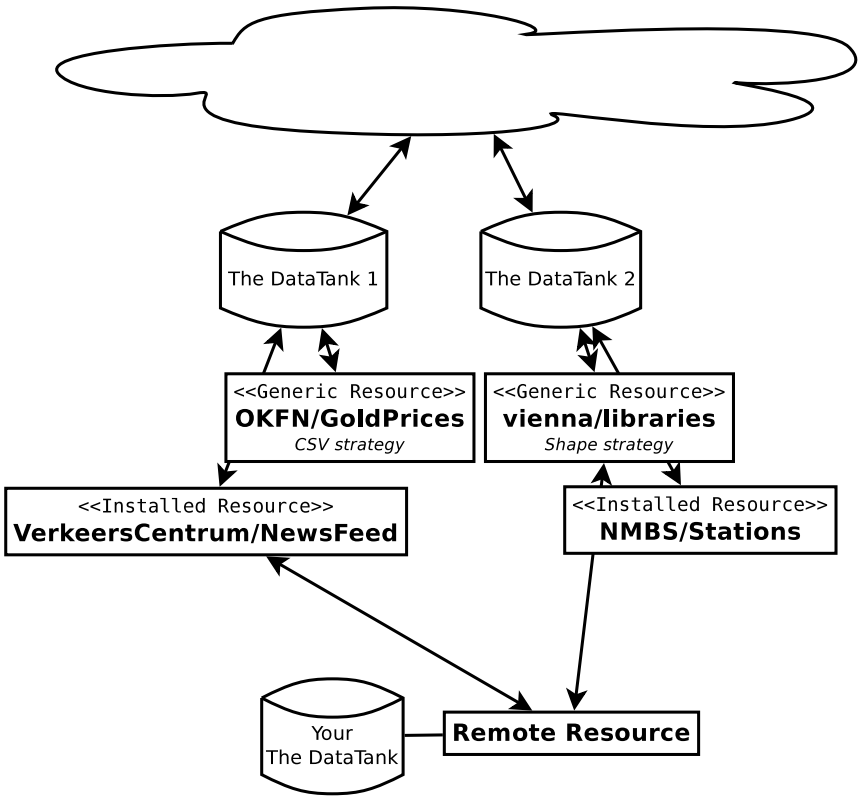
The DataTank wordt als gedistribueerd beschouwd. Hiermee wordt bedoeld dat elke instantie resources kan aanspreken die eigenlijk afgehandeld zullen worden door een andere The DataTank instantie. In dit geval is de resource een eenvoudige proxy die in het model kan worden aangesproken als eender welke andere (fig. 5.3).

5.3 URL-structuur

The DataTank kan al dan niet geïnstalleerd worden in een subdirectory of op een subdomain. Vanaf deze basis-URL kunnen alle resources benaderd worden:

5.3.1 Package/resource-paar

Het eerste wat we plaatsen is een package. Gaan we bijvoorbeeld naar de URL https://basisurl/package/, dan krijgen we een *directory listing* waarin alle resources in die package gegeven zullen worden. Als we een van deze resources eraan koppelen https://basisurl/package/resource, dan krijgen we de volledige inhoud van deze resource.



Figuur 5.3: Een voorbeeld van een The DataTank instantie die meerdere remote resources aanspreekt alsof ze bij deze instantie geïnstalleerd zijn.

5.3.2 Verplichte parameters

Een resource of strategy schrijver kan beslissen om verplichte parameters te verlangen. Als deze niet gegeven werden, dan krijgen we alweer een error met de volgende verplichte parameter die moet ingevuld worden. Dit wordt toegepast om bijvoorbeeld een route op te vragen bij iRail: `http://data.iRail.be/NMBS/Connections/Gent/Brussel/2012/01/23/12/00`. *Connections* stellen dan alle treinritten bij de NMBS voor, maar deze worden niet weergegeven aangezien dit onhaalbaar is. `NMBS/Connections/Gent` stelt alle ritten van de NMBS voor die vertrekken uit Gent. Ook dit is nog te veel en als we daar naar surfen, krijgen we een fout dat de parameter "destination" niet werd meegegeven. De volledige URL duidt een verzameling aan van verbindingen voor treinritten van de NMBS van Gent naar Brussel op een bepaald tijdstip.

5.3.3 REST-filters

Vooral om objecten te kunnen identificeren, hebben we ook REST-filters nodig. Deze gaan dieper in op een package/resource/verplichteparameters structuur. Zo zal `http://data.irail.be/NMBS/Connections/Gent/Brussel/` alle

connecties op dit ogenblik weergeven van Gent naar Brussel. Willen we nu van de eerste connectie de informatie over het vertrekstation, kunnen we dit makkelijk eruit filteren: `http://data.irail.be/NMBS/Connections/Gent/Brussel/0/departure/station`.

5.3.4 Open Search

Een simpele filter is mogelijk met behulp van de "Open Search"-specificatie. Hier kunnen we uit een array enkel die elementen opvragen die aan een bepaalde voorwaarde voldoen. Dit is heel beperkt maar is een standaard die wordt ondersteund door zoekmachines als die van *Google*. Een voorbeeld haalt er enkel de ritten uit met een duur van 2400 seconden:

```
http://data.irail.be/NMBS/Connections/
Gent/Brussel.about?filterBy=duration&filterValue=2400
```

5.4 Documentatie voor ontwikkelaars

Bij iedere implementatie van een resource hoort documentatie, alsook bij de gebruikte parameters of ingebouwde filters. De documentatie wordt automatisch gegenereerd in een eigen package, *TDTInfo* genaamd. Dit maakt de hele webservice zelfdocumenterend.

De browser richten naar `http(s)://uwhost/TDTInfo/Resources` geeft alle resources terug. Hier kunnen alle mogelijke formatters op worden toegepast, die op hun beurt beschreven staan in `TDTInfo/Formatters`.

Iedereen die op zijn of haar eigen host een The DataTank-instantie plaatst, kan op deze manier een grafische schil bouwen bovenop deze beschrijvende package en iedere gebruiker van deze web-service kan zonder veel voorkennis uitgebreid gebruikmaken van alle functies. Op het moment van schrijven zijn er zes resources in de package TDTInfo die een consument direct kunnen verder helpen.

5.4.1 Resources

Dit is een lijst van alle packages en resources, met hun verplichte parameters, hun niet verplichte parameters, creatiedatum, en documentatie. Ze bevat alle nodige informatie om eender welke resource aan te spreken. Ook de documentatie van zichzelf zit in deze resource. Zie codefragment 5.2.

```
TDTInfo: {
    creation_date: 1323012893,
    Resources: {
        doc: "This resource contains information
            about any resource in your TDT instance",
        requiredparameters: [ ],
        parameters: [ ],
        creation_timestamp: 1325333240,
        modification_timestamp: 1325333240
```

```
    },
    ...
}
```

Codefragment 5.2: Een JSON-weergave van de documentatie van TDTInfo zelf.

5.4.2 Admin

Generic resources en remote resources kunnen worden aangemaakt met behulp van een PUT call, mits authenticatie. Welke parameters daar moeten worden meegegeven voor iedere strategy, wordt hier gedocumenteerd.

5.4.3 Queries

Hoeveel vragen werden er iedere dag gesteld aan een bepaalde package/resource? Hoeveel foutmeldingen hebben we moeten weergeven?

5.4.4 Ontology

Aan welke ontologieën is de meegegeven package/resource gelinkt? Dit geeft triples terug.

5.4.5 Formatters

Welke formatters worden er ondersteund door deze The DataTank instantie?

5.4.6 Exceptions

Alle mogelijke foutmeldingen met hun HTTP foutcode.

Het is leuk om op te merken dat ook in elk van de TDTInfo functies de TDTInfo package zelf kan worden opgevraagd. Bijvoorbeeld: `http://data.irail.be/TDTInfo/Resources/TDTInfo/Resources`.

5.5 Een tweerichtingsverkeer

Zoals aangehaald in de rationale draait een opendatabeleid niet alleen rond de data zelf, maar veeleer rond de feedback en informatie die kan worden gehaald uit de gebruikers van de data. The DataTank ondersteunt drie vormen van feedback:

Correcties Als er een fout optreedt bij het ophalen van een bepaalde dataset zal er automatisch een foutbericht worden gegenereerd. De dataset-eigenaar wordt aldus onmiddellijk op de hoogte gesteld van dit probleem.

Wat als er echter foute informatie wordt gegeven? Bijvoorbeeld, er wordt een lijst van evenementen weergegeven waarvan een concert om 4 uur zou beginnen, maar dit wordt in de dataset aangekondigd om 3 uur. Een gebruiker van een app kan dan een POST-request terug sturen (of op een knop in de user-interface drukken die de POST-request zal versturen) waarin staat dat deze informatie fout is en moet gecorrigeerd worden naar opgegeven waarde.

Vind ik leuk Gebruikers van apps moeten heel snel een bepaald evenement of deel van de weergegeven informatie leuk kunnen vinden. Ook hier zal een POST-request op de specifieke URI (het real-world object) een "like"[1] implementeren.

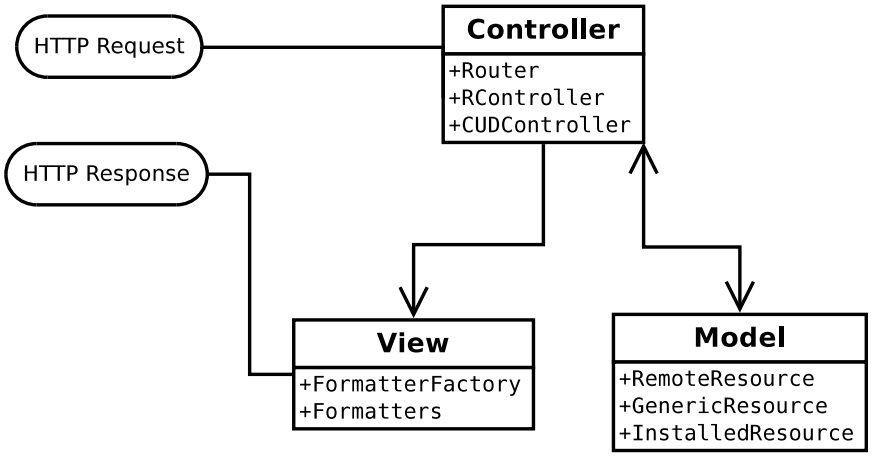
Commentaar Vaak wordt er op een bepaalde gebeurtenis commentaar gegeven. De mogelijkheid om commentaar toe te voegen wordt geïmplementeerd en gepersonaliseerd door de schrijver van een resource. Dit gaat via zelf gespecificeerde POST-requests.

[1] Een "like" is facebook-taal om op een simpele manier digitaal een blijk van appreciatie te geven.

Hoofdstuk 6

Architectuur van The DataTank

Om de architectuur te duiden maken we gebruik van een schema dat we verder stuk voor stuk zullen ontleden.



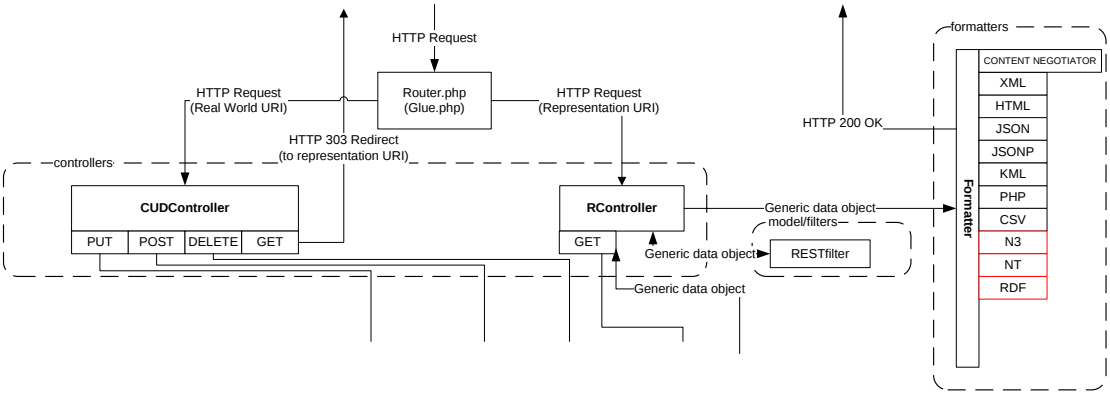
Figuur 6.1: Het MVC-model van The DataTank

The DataTank is geïnspireerd door het MVC-model, om model, representatie en controle van elkaar te scheiden.

6.1 Controller

Als een request, eender welke, binnenkomt op The DataTank, zal die passeren langs de router. Deze file zal de aanvraag ontleden en doorsturen naar de juiste controller. Router is hier de “single point of entry”.

We hebben twee controllers. In de router kan je naar hartelust zelf controllers bijvoegen als je een plug-in wilt schrijven die ook gebruik maakt van ons model



Figuur 6.2: De controllers en de views van The DataTank

en formatters. Dit zullen we doen om onze relationele interface in volgend hoofdstuk te verwezenlijken.

1. RController
2. CUDController

Om een real-world object te scheiden van z'n representatie hebben we twee controllers ingevoerd. Aan een real-world object kunnen we representationele data toevoegen. Op deze URIs zullen we dan ook de Create, Update en Delete commando's toelaten (vandaar de naamgeving). Een representatie kunnen we enkel lezen, daarom wordt de RController (Read) specifiek toegekend aan de representatie.

Bij de bespreking van beide controllers delen we alles op volgens 4 HTTP-methodes: GET, PUT, POST en DELETE. In de code kan je dit ook terugvinden als 4 te implementeren methodes in de abstracte controllerklasse.

6.1.1 RController

GET De GET-functie gaat sequentieel alles af wat moet gebeuren: formaat uitlezen, formatter opvragen, logging aansturen, het model om de juiste informatie vragen, enzovoort.

POST, PUT en DELETE Dit levert telkens een exceptie op. Een representatie kan je immers niet wijzigen. In codefragment 6.1 zien we wat er gebeurt.

```
pieterc@pieter:~$ curl -XPOST http://pieter.demo.thedatata
nk.com/thesis/diary.json
You cannot write to a representation.
Delete the format identifier (eg. .about or .json or .xml)
and try again
```

Codefragment 6.1: De exceptie zal resulteren in een HTTP 400 error

6.1.2 CUDController

GET Lezen van een real-world object is onmogelijk: een echt object door een internet-verbinding sleuren is nogal omslachtig. We kunnen ook hier een exceptie gooien. Dit zullen we niet doen om minder technische mensen niet te verwarren. In de plaats daarvan sturen we een HTTP 303 See Other bericht naar de .about formatter (zie codefragment 6.2) waar we bij de views dieper op in zullen gaan.

```
pieterc@pieter:~$ curl http://pieter.demo.thedatatank.com/t
hesis/diary/ -i
HTTP/1.1 303 See Other
Date: Sun, 18 Dec 2011 16:03:34 GMT
Server: Apache/2.2.20 (Ubuntu)
X-Powered-By: PHP/5.3.6-13ubuntu3.3
Location: http://pieter.demo.thedatatank.com/thesis/diary.a
bout
Vary: Accept-Encoding
Content-Length: 0
Content-Type: text/html; charset=UTF-8
```

Codefragment 6.2: De request zal resulteren in een 303 naar de .about pagina (zie views).

POST, PUT en DELETE Schrijven naar of verwijderen van een bepaalde resource wordt gedelegeerd naar het model. Uiteraard wordt er ook eerst gecontroleerd of de gebruiker het juiste username en wachtwoord meegeeft.

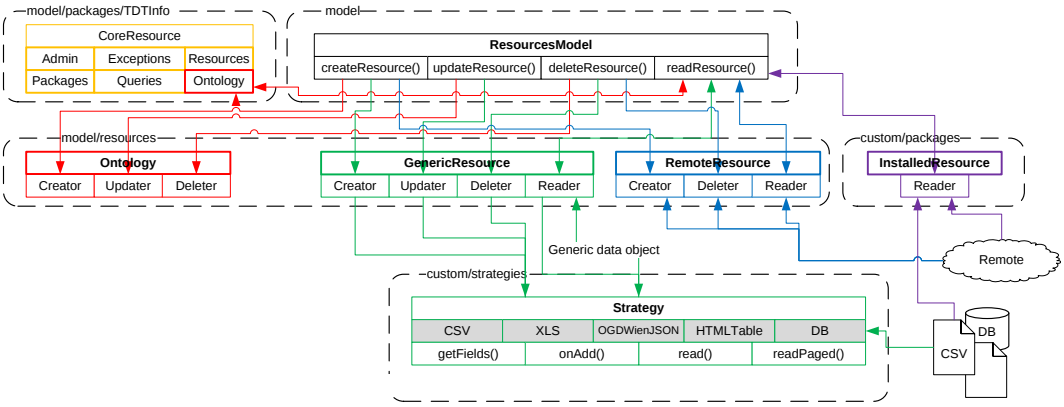
6.2 Views

Eenmaal het model een object genereerde, vraagt de controller de `FormatterFactory` om ons dat ene object te geven van een formatter. Een formatter is geschreven volgens het *template design pattern*. Eenmaal een formatter wordt gevraagd alles uit te printen, zal de abstracte klasse aan de afgeleide klasse vragen alle aparte stukken, van headers tot body, afhandelen.

Een formatter verwacht geconstrueerd te worden met een object als argument. Dit object is wat de controller terug zal krijgen van het model na de vraag van de gebruiker ontleed te hebben. Iedere formatter, eender welke, print het object naar een HTTP-response.

6.3 Model

Het model delen we op volgens de verschillende resources. Het `ResourcesModel` (zie figuur) zal dan volgens het abstract factory pattern optreden om de juiste reader, creator, updater of deleter aan te maken. Deze klassen ondersteunen voor elke soort resource (zie eerder) de 4 CRUD acties. Het ResourcesModel zal checken of alle verplichte parameters aanwezig zijn en de reader uitvoeren.



Figuur 6.3: Het model zonder het Semantische Web-gedeelte.

```
$model = ResourcesModel::getInstance();
$result = $model->readResource($packagename, $resourcename,
                               $parameters, $RESTfilters);
```

Codefragment 6.3: Een reader opvragen aan het model en het uitvoeren ervan.

Ter volledigheid geven we nog de uitgetekende versie van de volledige architectuur in figuur 6.4.

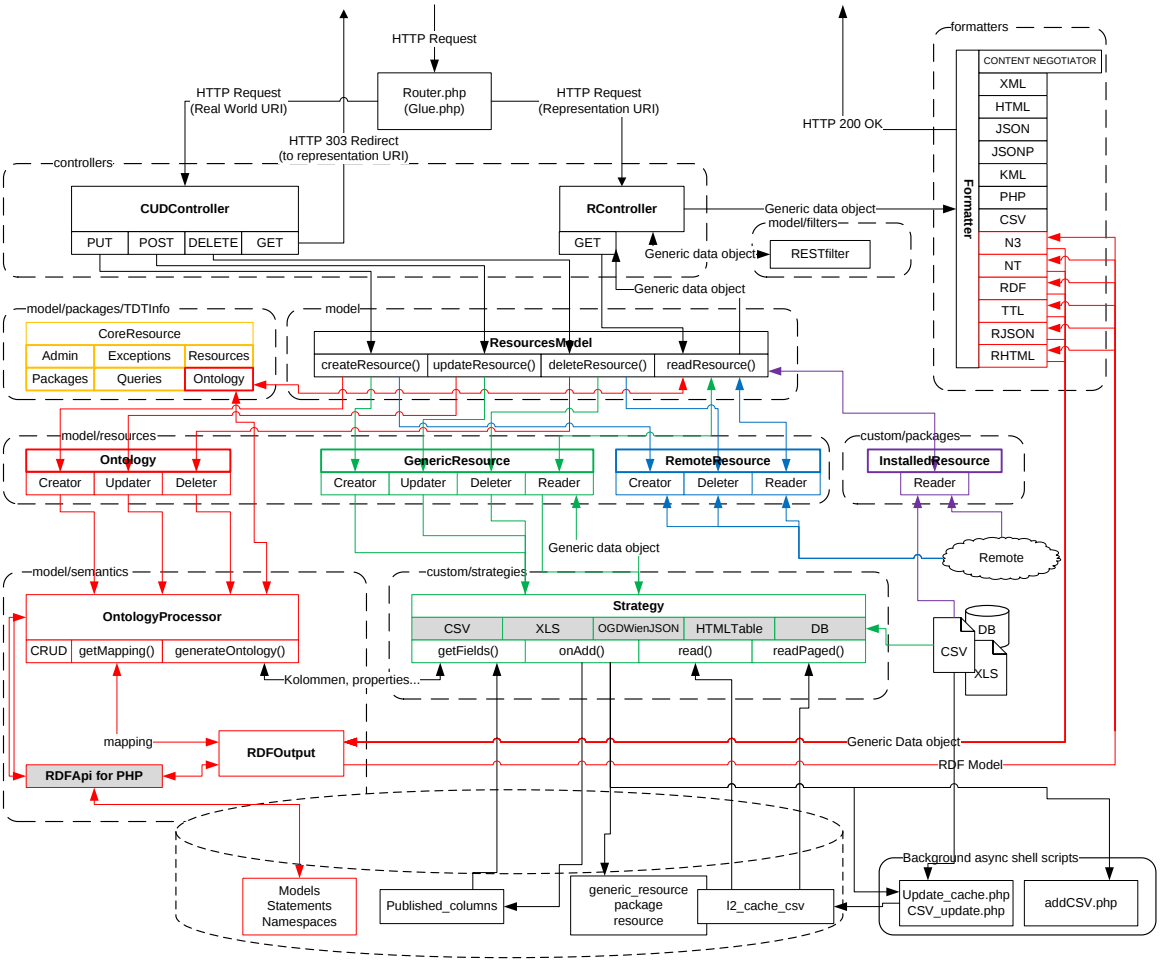
6.4 Eigen resources schrijven

Het enige wat nodig is om een eigen installed resource te schrijven, is het implementeren van een abstracte klasse `AResource`. De code moet vervolgens in `custom/package/packagenaam/resourcenaam.class.php` gezet worden.

Een eigen geschreven resource kan worden geregistreerd door hem toe te voegen in `custom/package/packagenaam/resources.php`.

6.5 Eigen strategies schrijven

Een strategy hoort bij een generic resource. Een strategy specificeert hoe een bepaalde externe resource aangesproken moet worden. Strategies schrijven wordt veelal als ingewikkelder ervaren omdat er ook een deel van de creator en deleter moet worden geschreven. Afhangende van de parameters die dan werden bijgeleverd wordt een andere resource ter beschikking gesteld aan het model.



Figuur 6.4: De architectuur van The DataTank getekend door Miel Vander Sande, MultiMediaLab UGent

Hoofdstuk 7

De relationele interface

Uit deel 1, de rationale, volgde dat er nood is aan een nieuw platform dat de overgang naar een beter opendatabeleid zou faciliteren. In het vorig hoofdstuk hebben we toegekeken op, en geholpen waar nodig aan, de implementatie van een platform dat een loutere data-adapter zou worden. Het model voorziet abstractie om makkelijk informatie op te vragen zonder enige kennis over waar de data zich werkelijk bevinden.

Wat heeft een dataconsument precies nodig? Kunnen we aan de hand van dit model een hulpmiddel bouwen waardoor dataconsumenten helemaal geen server-side programmatie meer moeten uitvoeren?

7.1 Waarom een simpele relationele interface?

Dataconsumenten kunnen eender wie zijn: mensen met een brede technische achtergrond, appdevelopers, data-journalisten, studenten, enzovoort. Als dataeigenaars consumenten willen stimuleren om zoveel mogelijk gebruik te maken van hun data, moet de drempel om ermee aan de slag te gaan zo laag mogelijk gehouden worden. Op onder andere "Apps for Ghent" werd het duidelijk dat de eerste stap die consumenten zetten eenmaal ze data in handen hebben, een web-service opzetten is, die de data omvormt tot een structuur die goed van pas komt bij hun specifieke use-case. Waarom bieden we dan zelf geen service aan die deze stap overbodig maakt? Zo hoeft de consument zelf geen webserver meer te onderhouden.

De oplossing moet op een gebruiksvriendelijke manier data kunnen omvormen tot een gewenste output. We onderscheiden deze basisfuncties:

- Selecteren
- Filteren
- Joinen/linken
- Sorteren

7.2 TDTQL

Deze functies moeten beschikbaar worden gesteld op zo'n eenvoudig mogelijke manier. De dataconsument moet bij eenvoudige opdrachten geen overbodige kennis opdoen, en met slechts enkele weetjes toch al heel veel kunnen doen. Hoe pakken we dit aan? We zullen een URL gebruiken om een query op te stellen.

De taal The DataTank Query Language (TDTQL) die uitgewerkt zal worden is sterk geïnspireerd door HTSQL [9] (zie rationale). Wat HTSQL vertaalt naar `WHERE`-clausules wordt nu vervangen door een package/resource paar binnen The DataTank. Vanaf een package/resource paar kunnen allerlei acties worden gelanceerd.

7.2.1 Een endpoint

HTSQL [9] start altijd vanaf een /, de root van een query, wat wij verder zullen aanzien als de eerste / na een basisurl. De gehele query kan meegegeven worden via de URL, wat een groot bijkomend gemak is voor een gebruiker: hij of zij kan een browser gebruiken om een query op te stellen, in tegenstelling tot POST- of PUT-requests, die een extra drempel hebben om testen toe te laten.

Net zoals SPARQL een endpoint aanbiedt, wordt het begin van een query als een endpoint op de basisurl gespecificeerd: https://basisurl/tdtql/.

Dit brengt uiteraard wat implicaties met zich mee. Een eerste is dat alle speciale karakters die worden gebruikt binnen de query moeten geëncodeerd worden in de url. Een spatie wordt dan bijvoorbeeld *%20*, een plus *%2B*, enzovoort. In de meeste talen zijn er echter functies voorzien op strings om dit op te vangen. De query hoeft dan enkel voor het concateneren met de basisurl even door deze encode-functie gehaald te worden.

Een tweede directe implicatie is dat er een mogelijke limiet komt te staan op de query, namelijk de lengte van een URL. RFC2616, de specificatie van HTTP, heeft het in geen geval over een maximumlengte. In een vergelijkende studie van boutell.com [23] vinden we meer informatie over wat momenteel gehanteerd wordt door diverse projecten:

- Microsoft Internet Explorer (Browser): 2048 karakters
- Microsoft IIS (Webserver): 13284 karakters
- Firefox (Browser): ongelimiteerd
- Apache (Webserver): 4000 karakters

Als conclusie trekt de auteur van het artikel dat een URL sowieso niet meer dan 2000 karakters zou mogen bevatten en dat dit een eerlijke bovenlimiet is. Dit geeft TDTQL nog steeds voldoende speelruimte.

7.2.2 Relationeel: best-effort

Binnen ons model in The DataTank is het makkelijk om data op te vragen, maar we zijn niet zeker in welke vorm we die ter beschikking krijgen. De eerste taak zal zijn om een best-effort relatie te zoeken binnen de verkregen dataset. Een relatie binnen TDTQL zal telkens worden gedefinieerd als een array binnen een array, waarvan de binnenste arrays associatieve arrays zijn. Een voorbeeld kan men vinden in codefragement 7.1.

```
$relatie = array(
    0 => array("sleutel1" => "waarde01",
               "sleutel2" => "waarde02",
               "sleutel3" => "waarde03"),
    1 => array("sleutel1" => "waarde11",
               "sleutel2" => "waarde12",
               "sleutel3" => "waarde13"),
    2 => array("sleutel1" => "waarde21",
               "sleutel2" => "waarde22",
               "sleutel3" => "waarde23"),
    3 => array("sleutel1" => "waarde31",
               "sleutel2" => "waarde32",
               "sleutel3" => "waarde33"));
```

Codefragment 7.1: Het model van een relatie gedefinieerd binnen The DataTank

Niet altijd zullen alle keys gegeven zijn en een value kan evengoed een object zijn of een nieuwe array met diepere elementen. Stel dat de waarden van de buitenste array telkens objecten zijn, kunnen we het object omzetten naar een associatieve array waarbij de publieke leden key-value-paren worden.

In de code zal er altijd moeten worden op toegezien dat alle randgevallen, mits deze inachtnemingen, behandeld worden.

7.2.3 De basisfuncties

Elk voorbeeld in dit hoofdstuk begint met /. Dit is de root van het TDTQL-endpoint. Voor iedere / kan dus dit worden toegevoegd:

```
http(s)://basisurl/tdtql
```

Selecteren

Selecteren wil zeggen dat we slechts bepaalde kolommen, of sleutels, zullen opvragen. We kunnen selecteren door na de identificatie van een bepaalde resource de accolades te openen en er een lijst aan mee te geven met sleutels.

```
/package/resource{sleutel1,sleutel2}
```

Om alles te selecteren kunnen we ook de wildcard “*” gebruiken.

Sorteren

Men kan de relatie sorteren op de sleutels die werden meegegeven, door er een + of een − achter te plaatsen. Een − zal de relatie aflopend sorteren, een + oplopend.

```
/package/resource{sleutel1+,sleutel2-}
```

Filteren

We kunnen ook slechts bepaalde zaken binnen een relatie weergeven op basis van verschillende filters. Nadat de resource werd opgegeven, kunnen we een ? plaatsen om een filterlijst te starten. In deze filterlijst kunnen sleutels worden opgegeven, alsook rekensommetjes worden meegegeven.

```
/package/resource?sleutel1>3.14159&sleutel2==2
```

Joinen/linken

Binnen een bepaalde selectielijst kunnen we een bepaalde sleutel linken aan een andere. Daarvoor introduceren we een nieuw teken:

```
/package/resource{*,sleutel1=>anderepackage/andereresource.anderesleutel}
```

Diverse

Het formaat wordt bepaald volgens de “HTTP Accept”-header. Als we deze content-negotiation willen overschrijven kunnen we gebruik maken van een “:format” operator op het einde van de query.

```
/package/resource...:xml
```

TDTQL ondersteunt slechts 2 types literals: numerieke en strings. Wil je specifiek een string meegeven, dan moet alles geplaatst worden binnen single quotes.

7.3 De parser

Een parser dient om een boom van opdrachten op te stellen en die vervolgens uit te voeren.

7.3.1 Backus-Naur Formaat

Backus-Naur Formaat (BNF) is een formaat om een grammatica of syntaxis te beschrijven. Het maakt gebruik van productieregels om de grammatica op te stellen. Een productieregel drukt telkens een substitutie uit.

Een parsergenerator kan vanuit een BNF een parser genereren. De werking van zo’n parsergenerator verschilt van product tot product.

```
input ::= segment END
segment ::= '/' ( top command* )?
command ::= '/' ':' identifier ( '/' top? | call | flow )?

top ::= flow ( direction | mapping )*
direction ::= '+' | '-'
mapping ::= ':' identifier ( flow | call )?

flow ::= disjunction ( sieve | quotient | selection )*
sieve ::= '?' disjunction
quotient ::= '^' disjunction
selection ::= selector ( '.' atom )*
...
```

Codefragment 7.2: HTSQL in BNF formaat ter illustratie

Als voorbeeld staat in codefragment 7.2 een stuk van de grammatica van HTSQL. Een ::= is een toekenning, een | is een of-teken.

7.3.2 Implementatie

Een parser schrijven valt uiteen in verschillende zaken:

- genereren van tokens uit de querystring,
- de tokens opslaan in een parsetree en
- de parsetree doorlopen en uitvoeren.

De tokenizer

De TDTQLTokenizer is een klasse die mits een opgegeven karakterlijst, de gehele string opsplitst in tokens. De tokens worden door de constructor opgeslagen als een private member lid. Met de functies `hasNext()` en `pop()` kan nu de lijst doorlopen worden om door te geven aan de parser zelf.

De beschikbare tokens worden gegeven in tabel 7.1. De rechterkolom zal worden gebruikt voor de grammatica van de parser.

Kiezen van een parsergenerator

Nu de aanvraag gesplitst werd in tokens, zal er een parse-tree worden opgesteld die elk query-onderdeel in de juiste volgorde kan uitvoeren. In dit hoofdstuk worden enkele parsergeneratoren besproken en wordt er beslist welke generator er zal gebruikt worden.

Karakter	gemapte waarde
.	‘.’
>	‘>’
<	‘<’
==	EQ
>=	GE
<=	LE
= !	NE
~	‘~’
(	‘(’
)	‘)’
{	‘{’
}	‘}’
=>	LN
|	‘|’
&	‘&’
!	‘!’
+	‘+’
-	‘-’
/	‘/’
*	‘*’
:	‘:’
,	‘,’
^	‘^’
%	‘%’
?	‘?’

Tabel 7.1: Alle mogelijke karakters binnen TDTQL tot nu toe gemapt op een waarde die kan verwerkt worden door de parser.

Bison Een heel logische keuze is om *GNU/Bison*, een project dat aan de hand van een *Yacc*-bestand een parser genereert, te gebruiken. Yacc is een formaat dat productieregels linkt aan stukken programmacode. Het grote voordeel aan Bison tegenover eender welke andere parsergenerator is dat Bison code genereert die onafhankelijk is van een bibliotheek. We moeten hier dus buiten de gegeneerde code, niets opnemen in ons eigen project.

Bison, aanvankelijk enkel voor C-talen, werd snel overgezet naar meerdere talen. Ook voor PHP is er momenteel een port in ontwikkeling. Aangezien deze software op het moment van schrijven in alpha-versie verkeerde, hebben we besloten onze oplossing elders te zoeken. [24]

Lemon Lemon is net zoals Bison een parsergenerator voor projecten in C. Het heeft echter enkele belangrijke verschillen:

- de grammaticastijl is anders waardoor typfouten minder voorkomen,
- de parser is thread-safe en
- wanneer we lemon code laten genereren, moeten we nog steeds het lemon-project opnemen in ons eigen project, wat integratie moeilijker maakt.

Dit laatste punt is ook een moeilijkheid als we nieuwe talen willen ondersteunen. De parsergenerator zelf heeft ook een nieuwe implementatie nodig in de taal die ze genereert.

Toch is het deze taal die een stabiele kloon heeft in PHP.

Lime Lime is een port van Lemon voor PHP. Ze gebruikt een grammatica als input, wordt gecompileerd naar een PHP-klasse en kan direct geïntegreerd worden binnen een bestaand project. Als library moet dan wel nog even een bestand worden geïncludeerd. Dit is de optie die we uiteindelijk gebruiken omdat het project al het meest matuur van de overlopen opties werd en makkelijk integreert met de codeerstijl van The DataTank.

De grammatica

De grammatica van HTSQL omzetten naar TDTQL zou te veel werk inboezemen. De grammatica van HTSQL was nog niet getest aangezien er geen gebruik werd gemaakt van een generator bij het programmeren van hun parser. Het zelf van scratch opbouwen van een Lemon-grammatica lijkt de beste optie. In codefragment 7.3 wordt een deeltje van de grammatica van TDTQL gegeven.

```
stmt = expression
| expression ':' format
.

expression = resource/$ '?' filterlist {
   $3->execute($$); }
.
```

```
resource = resourceid '{' selector '}' {
   $$ = $3->execute($1); }
| resourceid { $$ = $1->execute(); }
.

filterlist = filter { $$ = new TDTQLFilterList($1); }
| filterlist '&' filterlist {
   $1->merge($3,true); $$ = $1; }
| filterlist '|' filterlist {
   $1->merge($3, false); $$ = $1; }
| '(' filterlist/$ ')'
.

filter = name '>' string {
   $$ = new TDTQLFilter($1,">",$3); }
| name EQ string { $$ = new TDTQLFilter($1,"==",$3); }
| name '<' string { $$ = new TDTQLFilter($1,"<",$3); }
| name '~' string { $$ = new TDTQLFilter($1,"~",$3); }
| name '>' calc { $$ = new TDTQLFilter($1,">",$3); }
| name EQ calc { $$ = new TDTQLFilter($1,"==",$3); }
| name '<' calc { $$ = new TDTQLFilter($1,"<",$3); }
.

format = name {
   $ff = FormatterFactory::getInstance();
   $ff->setFormat($1); }
.
```

Codefragment 7.3: Een deel van de grammatica van TDTQL

De syntaxis van Lemon is relatief eenvoudig. Een element begint met een toekenning. Daarna volgt een lijst van karakters die in dat element zit. Daarna volgt eventueel een PHP-codefragement tussen accolades. De waarde van elk vorig element is dan telkens met een nummer bereikbaar.

Dit tdtql.lime-bestand moeten we compileren naar echte PHP code. Dit kan met de meegeleverde executable:

```
$ lime tdtql.lime > tdtql.class
```

tdtql.class voegen we vervolgens toe aan de include-lijst in ons project.

De klassenstructuur

We kunnen bij het opstellen van de parsetree niet onmiddellijk overgaan tot het uitvoeren van functies. Wel kunnen we aan de hand hiervan een post-fix-tree opstellen volgens het interpreter design pattern. Deze boom aflopen kan door simpelweg elke `execute`-functie op te roepen bij elk kind van een knoop.

In de root van de TDTQL-map kunnen we een README file vinden, heel wat klassen en nog 4 andere mappen.

calc/	parseexceptions.php	tdtql.class	TDTQLResource.class.php
filters/	README.md	tdtql.lime	TDTQLTokenizer.class.php
functions/	selectors/	TDTQLParser.class.php	TDTQLTools.class.php

Tabel 7.2: De TDTQL-parser mapstructuur geeft een overzicht van beschikbare klassen.

Parse-exceptions We definiëren onze eigen excepties binnen TDTQL zodat het duidelijk is wat nu in de fout ging en waar de fout zich precies bevindt. Als er zich een fout tijdens het parsen voordoet, zal een bericht aanduiden bij welk token het fout liep.

Er werden enkele excepties gedefinieerd. Alle werden ze afgeleid van AbstractTDTException.

- FunctionDoesNotExistTDTException: als een functie werd opgeroepen die niet bestaat.
- ParserTDTException: als de gebruikte query niet te verstaan is door de huidige grammatica.
- KeyDoesNotExistTDTException: als een sleutel werd opgevraagd die niet bestaat.

tdtql.lime en tdtql.class tdtql.lime is de file waarin we de grammatica specificeren. De .class file is de door lime gecompileerde file die bij het project gevoegd zal worden.

TDTQLParser De TDTQLController zal de query-lijn doorsturen naar de TDTQLParser, die het interpreter-pattern aan het werk zal zetten met de tokens verkregen van de TDTQLTokenizer.

TDTQLTokenizer Deelt de querystring op in aparte tokens. Dit gebeurt volgens de symbolen gedefinieerd in de TDTQLParser.

TDTQLTools Bevat hulpfuncties waar PHP tekortschiet. De enige functie die er nu in zit is catRelation. Deze zet 2 relaties die bij elkaar horen aan elkaar. Een functie die we veel zullen gebruiken bij het behandelen van een selectorlijst.

TDTQLResource Dit is wellicht de belangrijkste klasse. Het vraagt het model om een resource op te halen en deze om te zetten naar een relatie. Een relatie, zoals eerder gedefinieerd, bestaat uit associatieve arrays genest in een numerieke array.

Selectors

AArgument.class.php	TDTQLLink.class.php
TDTQLColumnName.class.php	TDTQLSelector.class.php
TDTQLWildcard.class.php	

Tabel 7.3: De map selectors bevat naast de selectorlijst code zelf, alle argumenten die een selectorlijst kan bevatten.

AArgument AArgument is de abstracte klasse die een argument van een selectorlijst beschrijft. Het enige wat deze klasse bijhoudt is een naam.

TDTQLSelector Dit selecteert data uit een resource met behulp van `{...}`. Het *tdtql.class*-bestand zal argumenten toevoegen aan de selector zolang er nog komma's zijn. Als `execute()` wordt opgeroepen worden eerst al de argumenten uitgevoerd. Als eer argument een + of een − had, dan wordt deze toegevoegd aan de "te sorteren"-lijst. Daar wordt pas rekening mee gehouden als alle argumenten uitgevoerd werden.

TDTQLWildcard Een wildcard ("*") selecteert alle kolomnamen uit de relatie en voegt ze toe aan het resultaat.

TDTQLColumnName Een kolomnaam zal de kolom toevoegen aan het resultaat.

TDTQLLink Een link, of `kolomnaam=>package/resource.anderekolomnaam`, zal eerst de nieuwe resource ophalen en in het geheugen laden. Vervolgens mapt het de juiste elementen op het resultaat.

Filters Er zijn slechts 2 klassen gedefinieerd bij de filters:

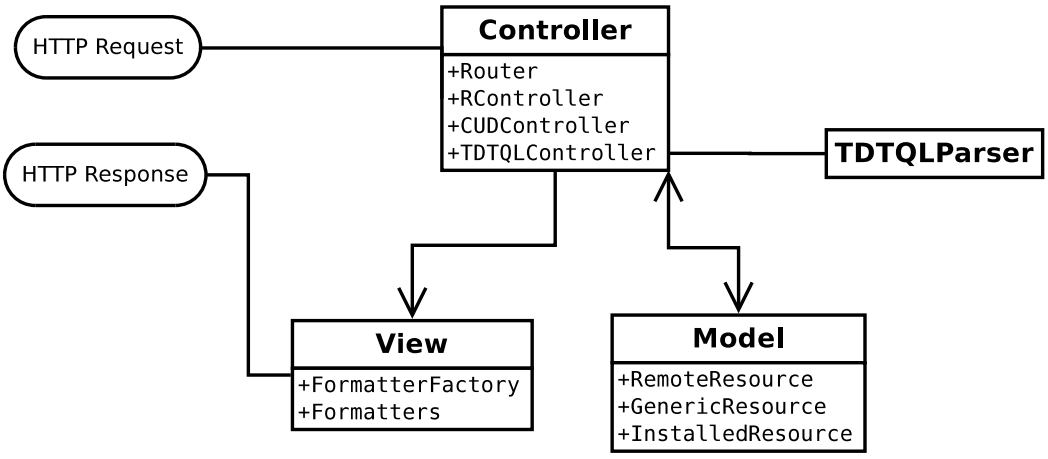
TDTQLFilter Deze klasse handelt de filter af. Ze bevat verschillende comparators. Als een element aan de voorwaarde van de comparator voldoet, wordt deze opgenomen in het resultaat.

TDTQLFilterList Een filterlijst bevat net zoals de naam doet vermoeden een lijst van filters. tdtql.class voegt zelf nieuwe objecten van het type FilterList toe aan de lijst als het een and- of or-operatie tegenkomt. Bij het uitvoeren zal er bij een and-operatie de doorsnede van de twee verzamelingen worden genomen (zie codevoorbeeld 7.4).

```
$this->filter->execute($currentRelation);
$this->addedFilterList->execute($currentRelation);
```

Codefragment 7.4: Een intersectie van twee relaties

Als er een or-operatie wordt verwerkt, zullen we de unie moeten nemen van beide verzamelingen zoals te zien is in codefragment **??**.



Figuur 7.1: Het MVC-model van The DataTank met TDTQL bijgevoegd

```
$result2 = $current; // copy first, we want to merge afterwards
$this->addedFilterList->execute($result2);
$this->filter->execute($currentRelation);
$currentRelation = array_merge($currentRelation,$result2);
```

Codefragment 7.5: Een unie van twee relaties

calc In calc komen de basis rekenkundige bewerkingen aan bod.

functions Deze directory bevat een FunctionFactory en een abstracte klasse voor de functies. Deze abstracte klasse is tevens afgeleid van AArgument. AArgument wordt uigebreid met een lijst van argumenten.

7.4 Integratie in het project

De integratie binnen de huidige codebase van The DataTank valt bijzonder makkelijk. Het model geeft ons voldoende abstractie om de juiste informatie zonder problemen op te vragen en ook een nieuwe hook maken om requests op te vangen is makkelijk.

Een nieuwe controller Aan de router voegen we een nieuwe controller toe: TDTQLController. Deze controller ondersteunt enkel GET-aanvragen en zal de parser aansturen. Dit wordt geïllustreerd in figuur 7.1.

De code van het TDTQL-endpoint staan enkel op de Git-repository van dit masterproefproject. Samenvoegen van de twee projecten vereist enkel kopiëren van onze repository in de main branch van The DataTank.

Deel III

Evaluatie

Hoofdstuk 8

Performantie-analyse

“

Meten is weten.

Anoniem

”

The DataTank doet uitstekend dienst voor kleine hoeveelheden real-time, dynamische of statische data. TDTQL voegt een extra laag toe die deze data beter zal structureren. Is er een vertraging merkbaar? Is een vertraging van The DataTank merkbaar tegenover het rechtstreeks scrapen van een databron, of net een versnelling? We doen een beknopt onderzoek.

Om de tests uit te voeren wordt Apache Bench (ab) gebruikt. Er zullen telkens in totaal 100 requests worden verstuurd. Alle tests werden gedaan via een 3.5G-netwerkverbinding.

HTTP-antwoordtijden In een boek van Jakob Nielsen, *Usability Engineering* [25], wordt een hoofdstuk gewijd aan antwoordtijden bij GUI's. Hij definieert dit alsvolgt (letterlijk citaat):

- *0.1 second* is about the limit for having the user feel that the system is reacting instantaneously, meaning that no special feedback is necessary except to display the result.
- *1.0 second* is about the limit for the user's flow of thought to stay uninterrupted, even though the user will notice the delay. Normally, no special feedback is necessary during delays of more than 0.1 but less than 1.0 second, but the user does lose the feeling of operating directly on the data.

- *10 seconds* is about the limit for keeping the user's attention focused on the dialogue. For longer delays, users will want to perform other tasks while waiting for the computer to finish, so they should be given feedback indicating when the computer expects to be done. Feedback during the delay is especially important if the response time is likely to be highly variable, since users will then not know what to expect.

Deze tijden zijn een goede maatstaf om in het achterhoofd te houden bij onderstaande tests.

8.1 TDTQL vs. REST

8.1.1 Opstelling

Het eerste test-scenario bestaat uit een dataset die op twee verschillende manieren wordt opgevraagd: de eerste keer via de gebruikelijke REST-interface, de tweede maal via het nieuwe TDTQL-endpoint.

```
$ ab -g rest.dat -n 100 \
http://data.irail.be/TDTInfo/Resources.json
```

```
$ ab -g tdtql.dat -n 100 \
http://data.irail.be/tdtql/TDTInfo/Resources:json
```

De resultaten werden nadien gesorteerd volgens hun tijdstip van uitvoeren:

```
$ (head -n 1 x.dat; tail -n +2 x.dat |\
sort) > xsorted.dat
```

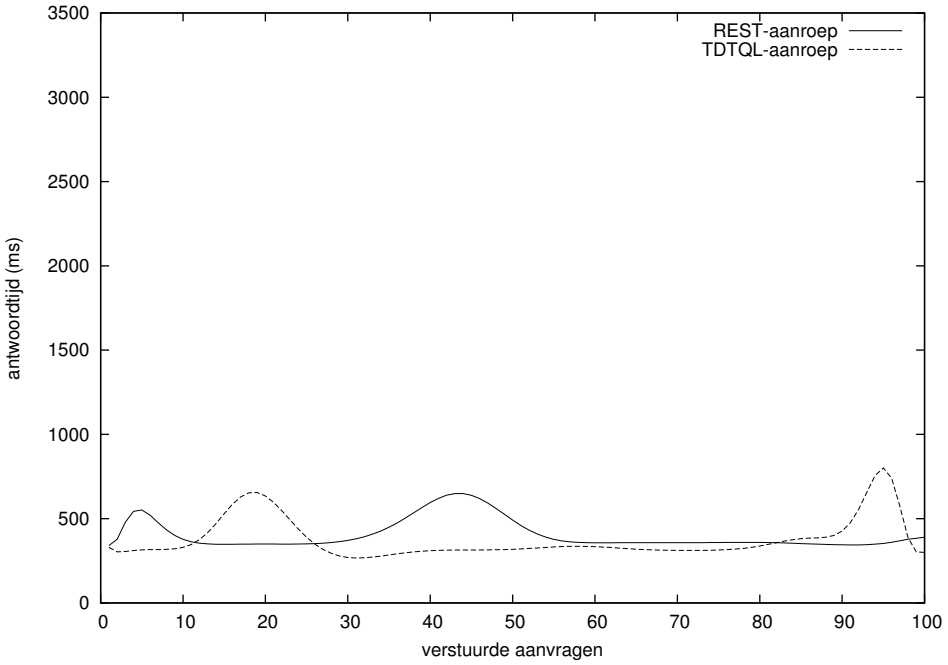
De `head` in dit bash-commando dient om de titelrij uit het bestand te abstraheren. De `tail` selecteert alle resultaten, zonder de titelrij en stuurt deze door naar het `sort`-commando. Aangezien het tijdstip vooraan iedere rij van dit bestand staat hoeft er niet gesorteerd te worden op een specifiek veld en zal dit commando geen verdere parameters verwachten.

Bij alle andere onderstaande testen werd dit ook uitgevoerd.

8.1.2 Het resultaat

In figuur 8.1 wordt de grafiek van het resultaat gegeven. Het gemiddelde, μ, is 330ms. De standaardafwijking, of σ, is $54,6$ms.

$\mu = 330$ms
$\sigma = 54,6$ms



Figuur 8.1: TDTQL vs. REST met een zelfbeschrijvende resource

8.1.3 Verklaring van het resultaat

Het resultaat kan tegenstrijdig lijken. De REST-interface doet er langer over om te antwoorden dan het TDTQL-endpoint. Nochtans doet deze laatste meer operaties op de dataset waarbij elke rij van de relatie wordt overlopen.

De verklaring moet gezocht worden bij de formatters en het verschil tussen objects en arrays. Het eerste wat de TDTQL-parser zal doen eenmaal het een resource tegenkomt, is die zo goed mogelijk omzetten naar een relatie. Een relatie, zoals eerder gedefinieerd, is, in de terminologie van TDTQL, een array van associatieve arrays, waar de associatieve arrays min of meer dezelfde sleutels bevatten.

In de test-cases haalden we telkens JSON op. Bij de REST-interface moet de json-formatter telkens een array van objecten afprinten. Bij de TDTQL-interface moet diezelfde json-formatter een array van arrays afprinten. Voor de ingebouwde PHP-functie json_encode geeft dit een opmerkelijk tijdsverschil dat ook te zien is in bovenstaande testen.

Af en toe verschijnen pieken in de grafiek. De oorzaak daarvan ligt bij de instabiliteit van het 3.5G-netwerk. De pieken zelf vallen nog steeds binnen een acceptabel tijdsinterval.

8.1.4 Conclusies

De antwoordtijd ligt tussen de 0,4 en 1 seconde. Door sommige datasets efficiënter aan te pakken kan de tijd met enkele ms dalen. In de toekomst zullen

de ontwikkelaars van The DataTank meer moeite moeten steken in het efficiënter schrijven van de formatters, aangezien deze een grote vertrager zijn gebleken.

8.2 Benchmark van uitgebreide queries

Om TDTQL zelf te benchmarken worden er telkens andere acties uitgevoerd op eenzelfde resource. De acties die besproken worden zijn sorteren, selecteren, filteren en linken.

In de grafiek van deze tweede benchmark staan er steeds 2 lijnen: de ene is de werkelijke test, de andere is een request op de resource zonder bijkomende acties.

Bij iedere opstelling wordt vertrek/aankomst-informatie van station Gent-Sint-Pieters opgevraagd. De body van het antwoord op deze query zonder bijkomende acties is 12KB groot, of 4077 karakters lang.

8.2.1 Selecteren

Opstelling

De eerste opstelling is een opstelling waarbij we enkele velden seleceren. TDTQL kopieert hierbij elke kolom van de relatie in een nieuwe array in het geheugen.

```
$ ab -g tdtql.dat -n 100 http://data.irail.be/tdtql/\
NMBS/Liveboard/Gent/2011/01/23/12/00/\
departures{time,iso8601,delay,direction,vehicle}:json
```

Resultaat

Het resultaat van deze benchmark staat in figuur 8.2. Het selecteren van slechts enkele attributen van een resource gaat gepaard met een lichte verbetering in snelheid.

$\mu = 365$ms
$\sigma = 70.4$ms

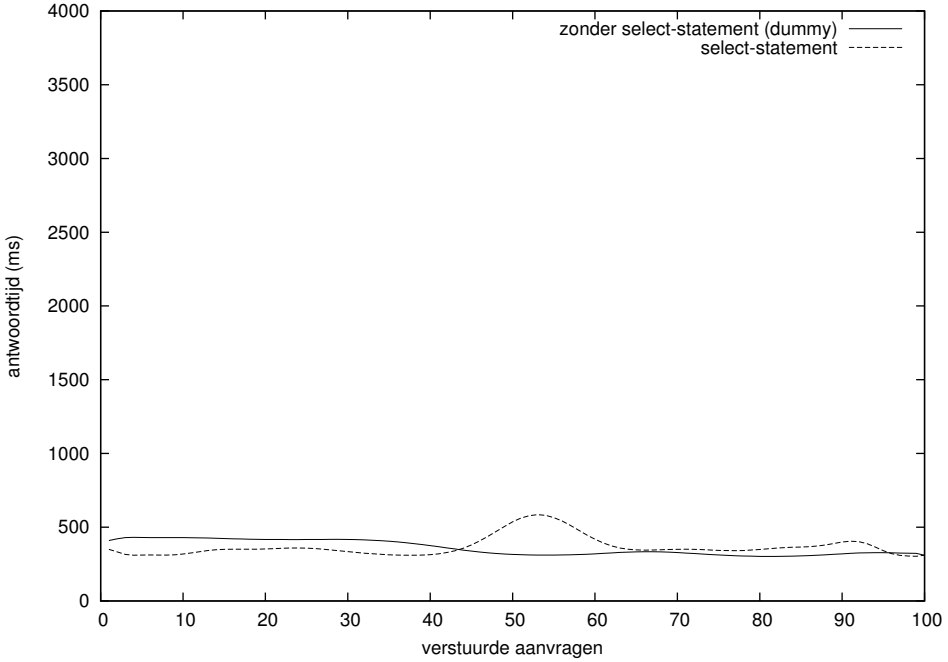
8.2.2 Sorteren

Opstelling

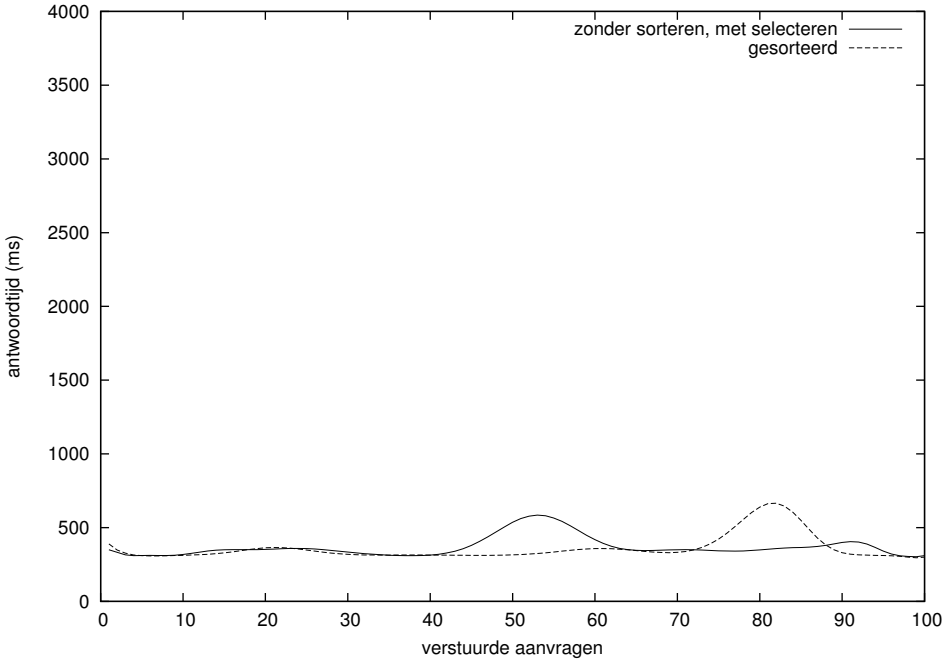
```
$ ab -g tdtql.dat -n 100 http://data.irail.be/tdtql/\
NMBS/Liveboard/Gent/2011/01/23/12/00\
departures{time,iso8601,delay-,direction-,vehicle-}:json
```

Resultaat

Het resultaat van deze benchmark staat in figuur 8.3. Het sorteren van een resource loopt ongeveer gelijk met het selecteren uit een uitkomst.



Figuur 8.2: Het selecteren tegenover niet selecteren van een resource



Figuur 8.3: Ophalen van een resource met selecteer- en sorteer-actie in contrast met de selecteer-actie

8.2.3 Filteren

Opstelling

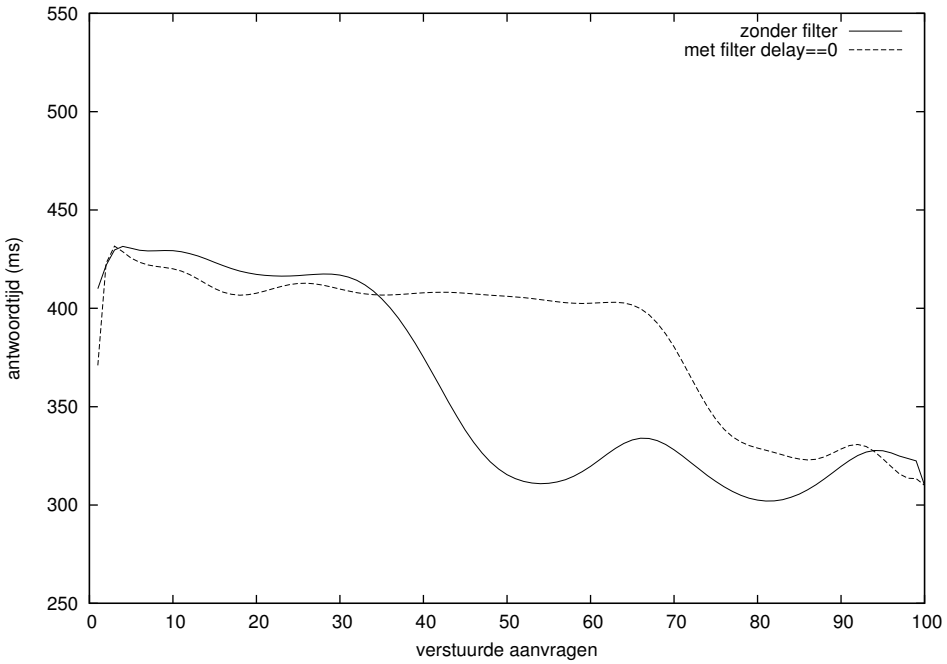
De derde opstelling is een opstelling waarbij een filter wordt toegevoegd. De resultaten van deze benchmark worden vergeleken met de dummy: een aanroep zonder bijkomende acties.

```
$ ab -g tdtql.dat -n 100 http://data.irail.be/tdtql/\
NMBS/Liveboard/Gent/2011/01/23/12/00/departures?delay==0:json
```

Resultaat

Het verloop van deze grafiek staat in figuur 8.4. Ook hier loopt de antwoordtijd ongeveer gelijk met de dummy, een request zonder acties.

$\mu = 385$ms
$\sigma = 45.9$ms



Figuur 8.4: Ophalen van een gefilterde resource

8.2.4 Linken

Opstelling

De laatste opstelling zal een link leggen tussen twee datasets. Als voorbeeld wordt hier weer uitgegaan van de vertrekken op 23 januari in Gent-Sint-Pieters

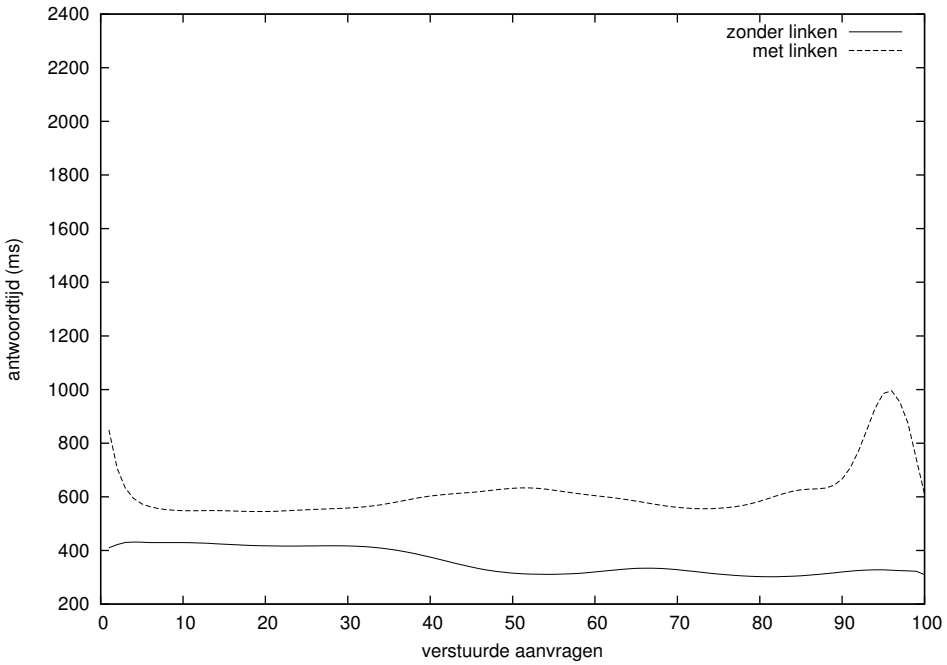
vanaf 12 uur (gedurende 1 uur). Iedere trein die vertrekt heeft een richting. Deze richting wordt gelinkt aan al de treinen die het volgend uur in dezelfde richting vertrekken. De output is een aggregatie.

```
$ ab -g tdtql.dat - n 100http://data.irail.be/tdtql/\
NMBS/Liveboard/Gent/2012/01/23/12/00/departures{\
direction=>NMBS/Liveboard/Gent/2012/01/23/13/00/\departures.direction\
,vehicle,delay,time}:json
```

Resultaat

Er worden sterke pieken waargenomen en een verdubbeling in antwoordtijd.

$\mu = 643$ms
$\sigma = 115.5$ms



Figuur 8.5: Het resultaat van een query met links in.

8.2.5 Conclusie

De extra wachttijd die sorteren, selecteren en filteren veroorzaken, is verwaarloosbaar. Het aggregeren van datasets blijkt echter een zware taak te zijn. Hoewel voor elke rij in de eerste relatie de tabel wordt geaggregeerd, blijkt de uitvoeringstijd slechts dubbel zo lang te zijn. Het grootste oponthoud is hier uiteraard het eenmaal ophalen van de tweede relatie.

Hoofdstuk 9

Gebruik en toekomst

“

Just because something doesn't do what you planned it to do, doesn't mean it's useless.

Thomas A. Edison,
Uitvinder en zakenman uit de 19e eeuw

”

Het is met The DataTank misschien iets minder uitgesproken als het met de uitvindingen van Thomas Edison was. Toch is het interessant om, eenmaal een platform werd opgeleverd, de mogelijkheden naar gebruik te herbekijken. Dit moet niet door de schrijvers van The DataTank zelf gedaan worden. Zij zitten immers vast aan de use-cases waar ze bij de start van de implementatie hun achterhoofd mee vulden. In het volgende hoofdstuk gaan we dieper in op andere organisaties die gebruik zullen maken van The DataTank. In het tweede hoofdstuk bespreken we de uitbreidmogelijkheden en de huidige beperkingen van TDTQL.

9.1 Gebruik

9.1.1 data.iRail.be

The DataTank wordt vandaag gebruikt als back-end API voor de iRail vzw. Aan deze The DataTank TDTQL toevoegen is een grote meerwaarde. Gebruikers kunnen nu real-time data over trein, tram, bus, wegen of vliegtuigen blootstellen aan hun queries.

Bijvoorbeeld alle treinen ophalen die nu vertrekken uit Gent die er vertraging hebben en daarop aflopend sorteren.

```
http://data.irail.be/tdtql/NMBS/Liveboard/
Gent Sint Pieters/departures{*,delay-}?delay>0
```

Ook met de vliegtuigen is een dergelijke query geheel analoog, wat de herbruikbaarheid van code doet toenemen.

```
http://data.irail.be/tdtql/Airports/Liveboard/
BRU/departures{*,delay-}?delay>0
```

BeLaws Binnen de iRail vzw is dit niet het enige waarvoor The DataTank gebruikt wordt. Een product dat binnen de iRail vzw werd ontwikkeld om snel het Staatsblad te kunnen doorzoeken is BeLaws. Op de site BeLaws.be kan iedereen het Staatsblad doorzoeken door een simpele zoekterm in te geven. Aangezien de vereniging om die reden een data back-end moesten bouwen werd ook snel een plain-old XML/JSON web-service op poten gezet. Momenteel in ontwikkeling is een nieuwe RESTful web-service, waarbij The DataTank de rol van adapter zal vervullen. Dankzij The DataTank kunnen datasets snel worden toegevoegd, een URI worden gegeven aan bepaalde wetten, gelinkt worden aan een ontologie, enzovoort.

iWay Een andere werkgroep binnen de iRail vzw werkt aan een API om realtime wegeninformatie aan te bieden. Deze maken, net zoals data.iRail.be, gebruik van webscrapers. Een eerste versie wordt verwacht voor het voorjaar van 2012.

Een enquête In 2012 wordt een enquête gehouden rond studentenmobiliteit in Vlaanderen en Brussel op `http://stumo.be`. De ruwe gegevens zullen via The DataTank ter beschikking worden gesteld van de vereniging en data-journalisten. TDTQL moet de mogelijkheid bieden aan minder technische mensen om toch bewerkingen op de dataset uit te voeren en zo eigen conclusies te trekken.

9.1.2 Apps for Ghent 2

Rekening houdend met de feedback die kwam van de eerste Apps for Ghent in 2011 organiseert de Stad Gent in samenwerking met het IBBT en Open Knowledge Foundation (OKFN) een tweede Apps for Ghent. Het format zal iets gewijzigd worden t.o.v. vorig jaar. Zo zal er bijvoorbeeld niet op de dag zelf geprogrammeerd worden. Twee maanden op voorhand wordt toegang gegeven tot een API met datasets van de Stad Gent. Deze API mag gebruikt worden door dataconsumenten die meedoen aan de Apps for Ghent wedstrijd. Op de dag zelf wordt de API publiek gesteld en de winnende toepassingen voorgesteld.

De API wordt uiteraard verzorgd door The DataTank. Later dat jaar volgen nog Apps for X events, waar X telkens een vakgebied of geografische locatie is.

9.1.3 Wenen

Wenen stelt zelf data ter beschikking op `http://data.wien.gv.at/`. Deze data zijn heel goed gestructureerd, maar worden helaas niet gelinkt aan andere datasets online, hebben geen feedback mogelijkheden en dergelijke meer. Om deze data in The DataTank te ondersteunen was er niet veel moeite nodig. Enkele PUT-requests brachten ons tot het resultaat op `http://datatank.demo.ibbt.be`.

Wenen onderzoekt nu zelf ook de mogelijkheid om samen met het team achter The DataTank hun data te verrijken [26].

9.1.4 Zeropoint.IT

Zeropoint.IT onderzoekt momenteel met enkele partners de mogelijkheid om een spin-off bedrijf te starten dat zich volledig toespitst op instanties te helpen een opendatabeleid te implementeren.

9.2 TDTQL nu en morgen

“

Release early, release often.

Eric S. Raymond,
Auteur van “The Cathedral and the Bazar”,
een geroemde essay over het opensource-onwikkelingsmodel

”

Na een jaar ontwikkelen, wikken en wegen, op- en onderzoeken wordt een verhaal afgeleverd met een open einde. We kunnen een eindbalans opmaken van wat er nooit mogelijk zal zijn, wat wel en wat de volgende stappen zijn.

9.2.1 Beperkingen

Ieder systeem heeft zijn beperkingen. Zo zal TDTQL nooit gebruikt worden voor, of TDTQL is alvast niet gebouwd op, het afhandelen van de zogenoemde “big data”. The DataTank handelt kleine datasets af die in-memory worden verwerkt. In het HTTP-antwoord bericht wordt onmiddellijk de response in een stuk meegegeven.

Een tweede belangrijke beperking is de aggregatiemogelijkheid van TDTQL. Het is slechts mogelijk een datasetelement te linken met een element uit een andere resource. We kunnen dus noch required parameters, noch verschillende resource-identifiers, als variabel veronderstellen in een link. Dit resulteert in $O(n)$[1] nieuwe requests, wat leidt tot een $O(n)$ keer langere uitvoeringstijd t.

[1] Met n het aantal elementen in de eerste relatie.

Omdat $n * t_0 > 3s$ vanaf $n \approx 5$ en met $t \approx 0,7$ (zie hoofdstuk benchmarks) aanzien we dit als onaanvaardbaar. Als we linken naar een element uit een andere resource hoeft er maar 1 extra opvraging aan het model gedaan te worden. Dit is dus $O(1)$ en onafhankelijk van hoeveel rijen er zijn[2].

9.2.2 Volgende stappen

In februari/maart 2012 zal TDTQL officieel worden geïntegreerd in de betaversie van The DataTank 2.0. Het is een opensourcetool die momenteel al kan gebruikt worden. Toch moet er nauw worden toegezien op een snelle releasecycle die de verdere ontwikkeling van TDTQL garandeert.

TDTQL is de huidige werknaam. Het zal wellicht echt het licht zien onder de naam Spectql Provides an Easy and Concise The datatank Query Language (SPECTQL), wat een knipoog is naar de veel bekendere taal SPARQL.

Een lange lijst...

- Momenteel kunnen relaties enkel gesorteerd worden op een natuurlijke manier (hoe mensen het zouden sorteren). Door toevoeging van s (string) of een n (number) na een + of een − zou ook specifiek op strings of numbers gesorteerd worden.

- Het is nu enkel mogelijk om te aggregeren/linken op gelijke sleutels. Het moet mogelijk worden, hoewel daar nog geen syntaxis voor is in HTSQL, om een voorwaarde voor het linken mee te geven.

- Ook als we aggregatiefuncties toepassen zouden we een window moeten kunnen meegeven. In de plaats dat we op heel de bijgevoegde array een operatie uitvoeren, kunnen we nu ook los van deze context werken. Een "count(argument over voorwaarde)" telt dan enkel in de context van de "over" voorwaarde.

- Sleutels die we als resultaat weergeven moeten een alias kunnen krijgen als de gebruiker die opgeeft. Zo lossen we het probleem op van dubbele kolomnamen en kunnen we ook betere namen geven aan geaggregeerde kolommen.

- Sommige queries gaan traag doordat veel werk telkens opnieuw moet worden gedaan. Net zoals bij SQL zouden we de mogelijkheid moeten aanbieden views en stored procedures aan te maken.

- Queries worden momenteel niet gecachet. Verschillende stukken van een query worden vaak herhaaldelijke keren gedaan en een query op zich wordt meestel veel na elkaar opgevraagd. Er moet worden onderzocht wat deze stukken precies zijn en hoe we het best queries kunnen cachen zodat de cache niet onmiddellijk overbelast wordt.

[2]We verwaarlozen hier de verwerkingstijd tegenover het opvragen en verwerken van een nieuwe databron.

- Aparte stukken objecten worden binnen de output van een query onduidelijk. We moeten immers op bepaalde objecten feedback kunnen geven. Om dit euvel te verhelpen kunnen we URI's afprinten bij iedere object waarop feedback kan gegeven worden.
- Momenteel zijn er enkel filters die filteren op een natuurlijke manier (hoe mensen het zouden doen: 'abc9' < 'abc10'). Er zijn nog heel wat andere filters mogelijk waaronder de geofilters. Filter bijvoorbeeld enkel die elementen waarvoor de locatie dichter is dan Xkm.
- Een filter die belangrijk genoeg is om in een apart puntje te vermelden: volgens ontologische informatie. Enkel elementen kunnen selecteren die bijvoorbeeld gelinkt zijn aan een "schema:Library" zou een handige toepassing zijn.
- Er kan slechts 1 niveau diep in de relatie worden geredeneerd. De puntoperator moet in de toekomst toelaten dat we bewerkingen doen met dieper gelegen structuren. Zo moet "count(platform.name)" voor hoeveel "platform"-elementen een naam gegeven werd.
- En nog veel meer...

Dankzij onze huidige architectuur is alles relatief snel implementeerbaar voor ontwikkelaars na deze masterproef. De student zelf werkt alvast in zijn vrije tijd door aan de verdere implementatie van TDTQL of SPECTQL.

Conclusie

Het doel van deze masterproef was de ontwikkeling van een module op *The DataTank*, teneinde ontwikkelaars toe te laten geschiktere uitvoer voor hun toepassing te verkrijgen uit meerdere databronnen door middel van een enkele oproep.

Een relationele interface bovenop The DataTank implementeren kwam gepaard met enkele vragen. Wat is een relationele interface en waartoe beperken we ons in deze scriptie? Hoe worden de databronnen geïdentificeerd? Hoe worden databronnen aangesproken?

Het antwoord op de eerste vraag, wat een relationele interface is en waartoe we ons beperken, werd beantwoord door de interface op te delen in verschillende acties. We beperkten ons tot *sorteren*, *linken*, *selecteren* en *filteren*. In de literatuursudie werd een querytaal besproken die deze functies al ondersteunde: SPARQL. Deze vereiste echter dat alle data beschikbaar waren in RDF-tripels. Aangezien het Web in 2012 nog niet semantisch genoeg is werd deze optie geschrapt. Omdat ook geen andere taal aan alle vereisten voldeed, hebben we zelf een querytaal gespecificeerd gebaseerd op HTSQL voor gebruikt binnen The DataTank. Oorspronkelijk TDTQL genoemd, later hernoemd naar SPECTQL, onderteunt deze querytaal nu de vier aangehaalde acties. Naderhand is gebleken dat *limiteren* hier het rijtje had kunnen aanvullen. In de volgende versie van SPECTQL zal dit er zeker aan toegevoegd worden.

Voor het antwoord op de tweede en derde vraag, hoe bronnen worden aangesproken en hoe ze worden geïdentificeerd, werd raad gezocht bij het HTTP-protocol, REST en het *HTTP Range 14 issue*. Met behulp van URI's en URL's kunnen bronnen respectievelijk worden geïdentificeerd en opgevraagd. Het eerste geeft een identificatie van een real-world object. Het tweede zal een representatie teruggeven van dit object. Als de URI wordt aangesproken zal de gebruiker geredirect worden naar de representatie URL die aan *content-negotiation* zal doen. SPECTQL zelf kan gezien worden als een nieuwe laag die van deze URI's gebruik maakt om query's uit te voeren. Een SPECTQL-query kan worden uitgevoerd door een HTTP GET aanvraag te versturen naar een SPECTQL-endpoint.

Er werd ook toegezien op de opbouw van de architectuur van The DataTank, opdat de relationele interface snel implementeerbaar zou zijn. Er werd gekozen voor een *MVC*-achtig model waarbij de *router* zal beslissen welke *controller* de

aanvraag moet afhandelen. De controller zal contact leggen met het model om de juiste dataset in het geheugen te laden. Dit vereenvoudigde het toevoegen van het SPECTQL-endpoint. Er werd een nieuwe controller aangemaakt die de query-string ontleedt door middel van een *parser*.

Handmatig een parser schrijven voor een nieuwe query-taal, hoe eenvoudig de taal ook mag zijn, is geen goed idee omdat de oplossing niet snel veranderbaar zal zijn. Met parsergeneratoren kunnen aan de hand van vooraf opgestelde grammatica's parsers gegenereerd worden. Er werden enkele generatoren uitgetest om uiteindelijk te kiezen voor *lime*, een PHP-kloon van lemon.

Als laatste deel van de masterproef werden de antwoordtijden van verschillende query's getest. Het algemeen resultaat is positief: de relationele interface levert geen noemenswaardige vertraging op tegenover de REST-interface. Een databron linken verdubbelt de wachttijd echter. Deze zwaardere operatie valt gelukkig nog steeds binnen een acceptabel interval. Uit de benchmarks leerden we dat de formatters slecht geïmplementeerd waren. In de toekomst zal er meer aandacht aan worden besteed.

Het linken van datasets kan momenteel slecht volgens de is-gelijk-aan-comparator. De syntax van HTSQL laat immers geen andere comparatoren toe. De verdere ontwikkelaars van SPECTQL zullen hier dus moeten afwijken van de HTSQL-specificatie.

De toekomst voor SPECTQL ziet er rooskleurig uit. Niet alleen wordt SPECTQL opgenomen in de volgende versie van The DataTank, ook heeft ze er enkele early-adopters bij die de taal verder zullen gebruiken.

Lijst van figuren

1 Onhoudbaar model voor mobiele toepassingen: de client moet te veel data binnenhalen 4

2 De toekomstige architectuur van The DataTank 4

1.1 Er zijn drie strategieën, al dan niet bewust, waarop een organisatie of bedrijf een meerwaarde kan halen uit datafeedback op dynamische en real-time data: crowd sourcing (fig. 1.1b), leren uit feedback (fig. 1.1c) en het analyseren van het gebruik (fig. 1.1d). 14

1.2 Open Government Data Camp 2011: een stukje van de groepsfoto 15

3.1 De CAP-stelling [10] 24

4.1 The Linked Open Data cloud in September 2011 (© w3c - Creative Commons By Sa: attribute and share alike) 37

5.1 Een voorbeeld van een geïnstalleerde resource in UML: VerkeersCentrum 42

5.2 Een voorbeeld van een generieke resource 43

5.3 Een voorbeeld van een The DataTank instantie die meerdere remote resources aanspreekt alsof ze bij deze instantie geïnstalleerd zijn. 44

6.1 Het MVC-model van The DataTank 48

6.2 De controllers en de views van The DataTank 49

6.3 Het model zonder het Semantische Web-gedeelte. 51

6.4 De architectuur van The DataTank getekend door Miel Vander Sande, MultiMediaLab UGent 52

7.1 Het MVC-model van The DataTank met TDTQL bijgevoegd . . 63

8.1 TDTQL vs. REST met een zelfbeschrijvende resource 67
8.2 Het selecteren tegenover niet selecteren van een resource 69
8.3 Ophalen van een resource met selecteer- en sorteer-actie in contrast met de selecteer-actie 69
8.4 Ophalen van een gefilterde resource 70
8.5 Het resultaat van een query met links in. 71

Codefragmenten

4.1 De broncode van de langst niet aangepaste webpagina op het Web: `http://www.w3.org/History/19921103-hypertext/hypertext/WWW/Link.html` - Tim Berners-Lee publiceerde deze op dinsdag 13 november 1990 15:17:00 GMT 29

4.2 Een voorbeeld van een HTML5-pagina 29

4.3 Een voorbeeld van een RDFa website die de DERI ontologieën gebruikt. 30

4.4 Een voorbeeld van open graph meta tags 31

4.5 Een voorbeeld van microformats. Meerbepaald een hcard. 32

4.6 Een voorbeeld van RDF/XML 33

4.7 Een SPARQL query om alle bibliotheken in Wenen op te halen en weer te geven. 34

4.8 Deze CSV-file wordt ingeladen op http://pieter.demo.thedatatank.com/thesis/diary 36

4.9 Deze resource staat live op http://pieter.demo.thedatatank.com/thesis/diary/0.json 37

5.1 Een voorbeeld van hoe de cache kan worden gebruikt. 41

5.2 Een JSON-weergave van de documentatie van TDTInfo zelf. . . . 45

6.1 De exceptie zal resulteren in een HTTP 400 error 49

6.2 De request zal resulteren in een 303 naar de .about pagina (zie views). 50

6.3 Een reader opvragen aan het model en het uitvoeren ervan. . . . 51

7.1 Het model van een relatie gedefinieerd binnen The DataTank . . 55

7.2 HTSQL in BNF formaat ter illustratie 57

7.3 Een deel van de grammatica van TDTQL 59

7.4 Een intersectie van twee relaties 62

7.5 Een unie van twee relaties 63

Verklarende Woordenlijst

API Application Programming Interface. 3, 5, 18, 20, 69

BNF Backus-Naur Formaat. 54

BSD Berkeley Software Distribution. 10

consument Dataconsumenten kunnen eender wie zijn: mensen met een brede technische achtergrond, appdevelopers, data-journalisten, studenten, enzovoort. Ze verwerken data om deze vervolgens aan te bieden aan een breder publiek als informatie.. 38, 44, 51

CRUD Create Read Update Delete. 35

CSV Comma-Separated Values. 37

EPSI European Public Sector Information. 11

EPSI Public Sector Information. 11

FSF Free Software Foundation. 10

GPL GNU General Public License. 10

GUI Graphical User Interface. 36, 63

HTML Hyper Text Markup Language. 29, 30, 32, 35

HTSQL Hyper Text Structured Query Language. 22, 52, 55, 57, 71, 73, 74

HTTP het HyperText Transfer Protocol. 22, 29, 32, 36, 37, 45, 48, 49, 52, 54, 70

IBBT Instituut voor BreedBandTechnologie. 2, 4, 18, 69

ISO International Organization for Standardization. 28

LODGE Linked Open Governmental Data Environment. 18

MIT Massachusetts Institute of Technology. 10

NoSQL NoSQL is een alternatief model om data voor te stellen. Het wordt gebruikt bij de sociale netwerksites twitter en facebook en stapt af van het relationeel model van Codd[8]. 23

ogdcamp Open Government Data Camp. 36

OGP ObjectGeörienteerd Programmeren. 27

OKFN Open Knowledge Foundation. 69

ontologie Een ontologie is meta-data. Ze beschrijft wat de eigenschappen zijn van een bepaald real-life object. Deze eigenschappen worden in in de representatie van een object terug verwacht. 27, 36

package Een DataTank package is een collectie van resources.. 44

RAM Random Access Memory. 23

RDBMS Relationeel Databank Management Systeem. 21, 22

RDF Resource Description Framework. 28, 29, 31, 33, 37, 73

RDFa Resource Description Framework in attributes. 30–32

real-world object Een real-world object wordt geïdentificeerd door een URI.. 27, 35–37, 47, 48

representatie Een representatie is een beschrijving van een real-world object. Een representatie is in termen van deze thesis een URI die gevolgd wordt door een extensie als .xml of .about. 27, 36, 37, 47, 48

resource Een DataTank resource is een end-point voor een bepaalde dataset. Resources worden onderverdeeld in packages. 36, 42–46, 49, 50, 53, 54, 60, 65

REST Representational State Transfer. 20, 22, 34, 35, 37, 38, 42, 64, 65, 69, 73, 74

RPC Remote Procedure Calls. 25

SOAP Simple Object Access Protocol. 34

SPARQL SPARQL Protocol and RDF Query Language. 33, 34, 37, 52, 71, 73

SPECTQL Spectql Provides an Easy and Concise The datatank Query Language. 71–74

TDTQL The DataTank Query Language. 52–54, 57–59, 61, 63–66, 68–73

UCB University of California Berkeley. 10

URI Uniform Resource Identifier. 8, 22, 27, 33, 35–37, 46, 47, 69, 72, 73

URL Uniform Resource Locator. 22, 32, 33, 35, 36, 42, 43, 52, 73

W3C World Wide Web Consortium. 26, 28, 29, 31, 33, 36

XML . 29, 30, 32, 33, 35

XSLT eXtensible Stylesheet Language Transformations. 32

YAML Yet Another MarkUp Language. 35

Literatuurlijst

[1] Pieter Colpaert en Miel Van der Sande. Follow the stars. http://www.slideshare.net/MielSande/the-datatank-25112011, 2011.

[2] Jason Kincaid. Yet another hot startup leaves a gaping security hole in its iphone app. http://techcrunch.com/2010/11/18/yet-another-hot-startup-leaves-a-gaping-security-hole-in-its-iphone-app/, November 2010.

[3] Blogger opent telenet digicorder api. http://www.tik.be/blogger-opent-telenet-digicorder-api, Juni 2010.

[4] Koen Baumers. Nmbs dwingt student te stoppen met routeplanner. http://www.standaard.be/artikel/detail.aspx?artikelid=GLF2RLI1K, Juni 2010.

[5] Tim Berners-Lee and Nigel Shadbolt. There's gold to be mined from all our data. *The Times, London*, December 2011.

[6] Robert Brauneis. Copyright and the world's most popular song. *GWU Legal Studies Research Paper No. 1111624*, page 69, October 2010.

[7] Karsten Lemmens. Eerste 'apps for ghent' pleit voor open data. http://www.standaard.be/Artikel/PrintArtikel.aspx?artikelId=IV3A3TCU, Mei 2011.

[8] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13:377–387, June 1970.

[9] Clark C. Evans. Htsql – a "nativ" web query language. PyCon, 2007.

[10] Hewitt, Eben. *Cassandra: The Definitive Guide*. O'Reilly, 2010.

[11] Eric A. Brewer. Towards robust distributed systems. In *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, PODC '00, pages 7–, New York, NY, USA, 2000. ACM.

[12] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33:51–59, June 2002.

[13] Noam Chomsky. *Aspects of the theory of syntax*. M.I.T. Press, Cambridge, 1965.

[14] *XML topic maps : creating and using topic maps for the Web*. Addison-Wesley, Boston, 2003.

[15] Ian Hickson. Html5 - a vocabulary and associated apis for html and xhtml. `http://dev.w3.org/html5/spec/Overview.html`.

[16] Austin Haugen. Abstract: The open graph protocol design decisions. In Peter F. Patel-Schneider, Yue Pan, Pascal Hitzler, Peter Mika, Lei Zhang 0007, Jeff Z. Pan, Ian Horrocks, and Birte Glimm, editors, *International Semantic Web Conference (2)*, volume 6497 of *Lecture Notes in Computer Science*, page 338. Springer, 2010.

[17] B. DuCharme. *Learning SPARQL*. O'Reilly Media, 2011.

[18] Eric Prud'hommeaux and Bristol Andy Seaborne, Hewlett-Packard Laboratories. Sparql query language for rdf. `http://www.w3.org/TR/rdf-sparql-query/`, January 2008.

[19] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Rfc 2616, hypertext transfer protocol – http/1.1, 1999.

[20] Michiel Gyssels. Masterproef: Studie naar een veilig en fraudeongevoelig framework voor het aanleveren van gepersonaliseerde web content, June 2011.

[21] W3C. Httprange14webography. `http://www.w3.org/wiki/HttpRange14Webography`, 2003.

[22] Danga interactive. Free & open source, high-performance, distributed memory object caching system. `http://memcached.org/`.

[23] Tom Boutell. Www faqs: What is the maximum length of a url? `http://www.boutell.com/newfaq/misc/urllength.html`, October 2006.

[24] Tim Landscheidt. De ontwikkeling van bison-php. `https://github.com/scfc/bison-php`.

[25] Jakob Nielsen. *Usability Engineering*. 1993.

[26] data.wien.gv.at. Datatank und semantifier. `http://data.wien.gv.at/apps/datatank.html`.

[27] Li Ding. *Enhancing Semantic Web Data Access*. PhD thesis, University of Maryland, Baltimore County, April 2006.

[28] Thijs Walcarius. Masterproef: Ontwerp en implementatie van een multimodale routeplanner met een mobiele reizigersassistent, June 2011.

[29] Alexandros Marinos, Erik Wilde, and Jiannan Lu. Http database connector (hdbc): Restful access to relational databases. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 1157–1158, New York, NY, USA, 2010. ACM.

[30] Glyn Moody. *Rebel Code: Linux and the open source revolution.* penguin books, 2001.

[31] Roy Thomas Fielding. *REST: Architectural Styles and the Design of Network-based Software Architectures.* Doctoral dissertation, University of California, Irvine, 2000.