

Toepassing van CQRS in combinatie met NoSQL

IN3405 - Bachelorproject 2012

Voor de Bachelor Technische Informatica, TU Delft

4-7-2012



Yorick Holkamp 1515632
Maarten van Meijeren 1526189

In opdracht van E.Novation B.V. en Technische Universiteit Delft, faculteit Technische Informatica.

Examencommissie

Bastiaan Bakker,
Jan Hidders,
Hans-Gerhard Gross,

Team Leader Development, E.Novation B.V.
Technische Universiteit Delft
Coördinator Bachelor, Technische Universiteit Delft

Voorwoord

Voor u ligt het eindverslag van de onderzoeksstage die heeft plaatsgevonden in de periode maart tot juli 2012 bij E.Novation te Capelle aan den IJssel. Deze stage is uitgevoerd in het kader en het eindproject voor de bachelor Technische Informatica gegeven aan de Technische Universiteit Delft.

Dit eindverslag bevat het beknopte resultaat van de verschillende documenten die tijdens het project zijn opgesteld. De eerder opgeleverde documenten zijn bij dit rapport gevoegd als bijlage voor de geïnteresseerde lezer.

We nodigen alle lezers uit om kennis te nemen van het hoofdstuk 3 ‘Probleemstelling’ van dit verslag. Dit hoofdstuk bevat de kern van ons project en schetst daarmee een aardige blik op de uitgevoerde werkzaamheden.

De lezer die een keuze voor een NoSQL database wil maken raden wij aan om hoofdstuk 4, 7 en bijlage D te lezen. In deze onderdelen van het rapport wordt een afweging gemaakt tussen de verschillende NoSQL databases en een afweging met betrekking tot het gebruik hiervan gemaakt.

Als een na laatste punt raden we de lezer met interesse voor het gebruik van CQRS aan om hoofdstuk 5 en 6 door te lezen, hier worden de voor- en nadelen en de implicaties van het gebruik van CQRS besproken.

We willen graag onze begeleider bij E.Novation, Bastiaan Bakker bedanken voor het mogelijk maken van deze stage en de geboden flexibiliteit in de uitvoering hiervan. Verder gaat onze dank uit naar Jan Hidders voor zijn rol als begeleider vanuit de Technische Universiteit Delft en zijn hulp met het afbakenen en vormgeven van onze opdracht. Ook bedanken we via deze weg graag de andere collega's van E.Novation die ons hebben bijgestaan en voor interessante gespreksstof hebben gezorgd in de kantine.

Samenvatting

In dit onderzoek hebben we de voor- en nadelen van het gebruik van NoSQL databases en het Command Query Responsibility Segregation(CQRS) met Event Sourcing(ES) pattern onderzocht.

In het onderzoek dat ingaat op de verschillende NoSQL databases zijn een groot aantal databases geselecteerd voor nader onderzoek. De vergelijking is gemaakt op basis van de kwaliteitsattributen relevant voor E.Novation. Uit dit onderzoek is MongoDB als beste NoSQL database naar voren gekomen.

Aan de hand van de twee verschillende projecten, waarbij de CQRS/ES variant gebruik maakt van MongoDB en de ander gebaseerd op een LA met een relationele database, is een vergelijkend onderzoek gedaan. In dit onderzoek is er geen significant verschil in ontwikkeltijd geconstateerd. Bij een complexer systeem zal het toevoegen van functionaliteit waarschijnlijk minder tijd kosten in een CQRS/ES systeem. De complexiteit van het CQRS/ES systeem is naar verwachting namelijk lager, waarbij er eenvoudig nieuwe componenten op de command- en eventbus aangesloten kunnen worden zonder kennis van de rest van het systeem.

Theoretisch zitten er veel voordelen aan het gebruik van MongoDB binnen een CQRS/ES systeem. MongoDB is een flexibele database, daaruit ontstaan verschillende voordelen specifiek voor de werking van CQRS/ES.

De keuze voor CQRS/ES met MongoDB heeft enige gevolgen voor de ontwikkeling en het onderhoud van een applicatie. Zo zal er door de ontwikkelaars kennis opgedaan moeten worden over deze systemen voordat er effectief mee ontwikkeld kan worden en zullen beheerders kennis op moeten doen om de beheertaken uit te kunnen voeren op de gewijzigde systemen.

De afweging of CQRS/ES en MongoDB een geschikte keus is hangt sterk af van de specifieke applicatie. Wanneer de te bouwen applicatie goed aansluit op de event-driven werkwijze van het Axon Framework zal het meerwaarde kunnen bieden. Indien dit niet direct het geval is zal een meer traditionele opzet meer geschikt zijn.

Woordenlijst

Dit hoofdstuk geeft een overzicht van de veelgebruikte terminologie in dit bacheloreindverslag.

Apache Maven - Tool om Javaprojecten te beheren en op automatische wijze te compileren.

BSON - Binary JSON, de binary tegenhanger van het JSON-formaat, dit formaat wordt onder andere intern gebruikt door MongoDB.

Checkstyle - Applicatie voor statische code-analyse welke controleert of Java-code voldoet aan regels omtrent stijl.

Command Query Responsibility Segregation - Pattern waarbij de afhandeling van aanpassingen aan het systeem (commands) losgekoppeld is van de presentatie van data.

CRUD - Create, read, update, delete, de basisacties die op een database uitgevoerd kunnen worden.

CQRS - Zie Command Query Responsibility Segregation.

CQRS/ES - De combinatie van zowel het Command Query Responsibility Segregation- als Event Sourcing-patterns. Mogelijk geïmplementeerd door het Axon Framework.

Cyclomatic complexity - Software metric welke een indicatie geeft van de complexiteit van een stuk software op basis van de hoeveelheid verschillende berekenpaden in het systeem.

Design pattern - Een patroon om een softwaresysteem in te delen om daarmee een veelvoorkomend probleem op te lossen.

EDA - Zie Event Driven Architecture.

Event Driven Architecture - Een design pattern gericht op het gebruik van events om de communicatie tussen systeemonderdelen te verzorgen.

Event Sourcing of ES - Pattern waarbij objecten niet in huidige staat worden opgeslagen in een databasesysteem maar enkel een serie events wordt opgeslagen welke, na het opnieuw afvuren van de events, tot het object in de huidige staat leiden.

Layered Architecture of LA - Een design pattern voor software waarbij er meerdere abstractielagen binnen een applicatie worden ingebouwd waarmee

MongoDB - Een open source document-georiënteerde database.

Nexus - Systeem om Maven repositories mee te beheren en een lokale kopie van software van derden aan te bieden op gecontroleerde wijze.

NoSQL - Een klasse databasesystemen die een andere aanpak hanteren dan relationele databases.

Spring MVC - Een open source framework om de bouw van web-applicaties te faciliteren.

1 Inhoud

Voorwoord	2
Samenvatting	3
Woordenlijst	4
1 Inhoud	5
2 Inleiding	7
3 Probleemstelling	8
3.1 Doelstelling	8
3.2 Opdrachtformulering	8
3.3 Onderzoeksvragen	9
4 Onderzoek NoSQL databases	9
4.1 Voorselectie	9
4.2 Uitgebreide analyse	10
4.3 Conclusie	10
5 Voor- en nadelen gebruik NoSQL	11
5.1 MongoDB als event store	11
5.2 MongoDB als query-database	12
5.3 Toepassingen met en zonder profijt	12
6 Implicaties applicatieontwikkeling	14
6.1 Wijzigingen ontwikkelproces	14
6.2 Systeemwijzigingen	14
6.3 Veranderingen applicatiebeheer	15
6.4 Efficiëntie ontwikkeling	16
7 Performance MongoDB	17
8 Proces	17
8.1 Initiële planning	17
8.2 Uiteindelijke projectplanning	17
8.3 Samenwerking	21
8.4 Begeleiding	22
8.5 Ervaringen op het bedrijf	23
8.6 Conclusie	24
9 Ontwerp	25
9.1 Algemeen ontwerp	25
9.2 Opbouw CQRS	26
9.3 Opbouw LA	28
10 Implementatie	29
10.1 Algemene implementatie	29
10.2 Implementatie CQRS	29
10.3 Implementatie LA	31
10.4 Testen	32
10.5 Kwaliteitsoordeel SIG	32
11 Conclusie	33
11.1 Doelstelling	33
11.2 Onderzoeksvragen	33
12 Aanbevelingen	35
12.1 Verbeteringen	35
12.2 Vervolgonderzoek	35

13	Reflectie	36
13.1	Algemeen	36
13.2	Maarten van Meijeren	36
13.3	Yorick Holkamp	37
14	Literatuurlijst.....	38
15	Bijlagen.....	39
15.1	A. Opdrachtschrijving	39
15.2	B. Oriëntatieverslag	39
15.3	C. Plan van aanpak	39
15.4	D. Research Document NoSQL Databases	39
15.5	E. Implementatieanalyse.....	39
15.6	F. Gebruik van Axon en MongoDB	39

2 Inleiding

Met de opkomst van bedrijven als Google, Twitter en Facebook die met een ander eisenpakket naar een databasesysteem kijken dan veel gevestigde bedrijven in de IT-sector heeft het 'NoSQL'-fenomeen een plekje veroverd in de wereld van grote namen als SQL Server, MySQL en PostgreSQL. Zo heeft Facebook Cassandra ontwikkeld om aan hun eisen voor de opslag en het doorzoeken van berichten tussen gebruikers te voldoen¹, wat later ook door bedrijven als Twitter is opgepakt².

Hoewel veel van de succesverhalen afkomstig zijn van scenario's waarin er met zeer grote hoeveelheden data gewerkt wordt komt de vraag naar voren of het gebruik van een NoSQL databasesysteem ook interessant is op een beperktere schaal. Binnen E.Novation is hierbij de nieuwsgierigheid ontstaan naar de vraag of het meerwaarde biedt om van de huidige databases over te stappen op een variant van NoSQL eventueel in combinatie met het toepassen van de design patterns Command Query Responsibility Segregation (CQRS) en Event Sourcing (ES). Dit project heeft geleid tot het project waarvan het eindverslag voor u ligt.

De probleemstelling waarbij hier mee is gewerkt sluit aan op de doelstelling om ervaring met deze zaken op te doen en luidt: "Hoe kan er binnen E.Novation gewerkt worden met NoSQL databases binnen applicaties ontwikkeld met de CQRS en ES design patterns?"

Om deze vraag te beantwoorden is er eerst een literatuuronderzoek uitgevoerd waarbij gekeken is naar de verschillende NoSQL databases die beschikbaar zijn om vervolgens een best passende oplossing te kunnen kiezen aan de hand van het eisenpakket. Vervolgens zijn er twee applicaties ontwikkeld, een op de 'standaard' wijze en een op basis van het CQRS en Event Sourcing pattern en een NoSQL database. Deze twee applicaties zijn functioneel identiek aan elkaar ontwikkeld om op deze wijze voor- en nadelen behorende bij de technologiekeuze inzichtelijk te maken.

In dit verslag zal als eerste stilgestaan worden bij de probleemstelling van het project waarna achtereenvolgens de verschillende deelvragen besproken zullen worden in de hoofdstukken 2 tot 5. Nadat in deze hoofdstukken de onderzoeksresultaten besproken zijn zal het proces tijdens het project worden besproken en het ontwerp en de implementatie van de twee applicaties toegelicht worden. Vervolgens zal er een conclusie aan het project verbonden worden en geven we enkele aanbevelingen. Tot slot zal er in de reflectie een terugblik op het project gegeven worden met daarbij onze ervaringen.

¹ A. Lakshman, 'Cassandra – A structured storage system on a P2P Network', 25 augustus 2008, http://www.facebook.com/note.php?note_id=24413138919

² A. Popescu, 'Cassandra @ Twitter: An Interview with Ryan King', 23 februari 2010, <http://nosql.mypopescu.com/post/407159447/cassandra-twitter-an-interview-with-ryan-king>

3 Probleemstelling

E.Novation vraagt zich af hoe er met de combinatie van NoSQL en CQRS/ES binnen het bedrijf gewerkt kan worden. Achter deze vraag zit een doel, waar een opdrachtformulering aan is gekoppeld. Deze opdrachtformulering heeft vervolgens geleid tot de probleemstelling met de daarbij behorende onderzoeksvragen.

3.1 Doelstelling

E.Novation wil kennis opdoen omtrent het gebruik van NoSQL-databases om zo bij toekomstige projecten een geïnformeerde beslissing te maken om deze technologie al dan niet toe te passen. Momenteel is er nog geen keuze gemaakt voor een specifiek NoSQL databasesysteem. Om tot een keuze te komen zullen wij een literatuuronderzoek uitvoeren en de beschikbare systemen toetsen aan de hand van de benodigde functionaliteit en gehanteerde kwaliteitseisen van het bedrijf.

Nadat bekend is welke DBMS het best aansluit op de kwaliteitseisen van E.Novation zal er een demoproject gemaakt worden om ontwikkelaars kennis met deze toepassing te laten maken met de toepassing van NoSQL in combinatie met CQRS en Event Sourcing. Hierbij kan er tevens een ‘sandbox’ aangeboden worden voor eventuele toekomstige proof of concepts.

3.2 Opdrachtformulering

Het onderzoeken van de toepassingsmogelijkheden van NoSQL databases binnen de huidige set van SaaS-diensten in het portfolio van E.Novation. In de oriëntatiefase worden de beschikbare NoSQL oplossingen aan de hand van een literatuurstudie vergeleken op basis van relevante functionaliteit, externe kwaliteitsattributen zoals robuustheid, prestaties, schaalbaarheid en interne kwaliteitsattributen zoals analyseerbaarheid, aanpasbaarheid, testbaarheid.

Vervolgens wordt een proof-of-concept uitgewerkt op basis van een van de bestaande toepassingen van E.Novation of een representatief fictief voorbeeld, waarbij gebruik gemaakt wordt van het Command Query Segregation (CQRS) pattern in combinatie met Event Sourcing. Hieraan zullen diverse tests worden onderworpen om bovengenoemde kwaliteitsattributen in de praktijk te meten. Daarnaast zal een statische kwaliteitsanalyse plaatsvinden (eventueel deels via de Software Improvement Group).

3.3 Onderzoeksvragen

Om aan de gestelde doelstelling te kunnen voldoen hebben we tijdens ons project de volgende probleemstelling gehanteerd: “Hoe kan er binnen E.Novation gewerkt worden met NoSQL databases binnen applicaties ontwikkeld met de CQRS en ES design patterns?”

Om deze probleemstelling te kunnen beantwoorden zijn de volgende deelvragen opgesteld en onderzocht:

1. Welke NoSQL-database sluit het beste aan op de interne en externe kwaliteitseisen?
2. Welke mogelijkheden en limitaties geeft een NoSQL database ten opzichte van de huidige SQL-oplossing?
 - a. Welke voordelen geeft de toepassing van NoSQL als Event Store binnen ES?
 - b. Welke voordelen geeft de toepassing van NoSQL als view database?
 - c. Welke toepassingen profiteren weinig of niet van een NoSQL-oplossing?
3. Wat zijn de gevolgen voor het ontwikkelproces van E.Novation?
 - a. Moet het ontwikkelproces gewijzigd worden?
 - b. Hoe kunnen systeemwijzigingen op later moment aan doorgevoerd worden?
 - c. In hoeverre moeten de applicatiebeheerders opgeleid worden?
 - d. Verhoogt het gebruik van CQRS/ES de efficiëntie van de ontwikkeling?
4. Wat zijn de performance-grenzen wanneer er gewerkt wordt met een NoSQL database?
 - a. Hoe ver reikt de performance van een enkele server?
 - b. Hoeveel winst levert het toevoegen van extra servers aan de database cluster?

De resultaten van het onderzoek naar de verschillende onderzoeksvragen zullen in de volgende hoofdstukken beschreven worden.

4 Onderzoek NoSQL databases

Om een passende NoSQL database voor E.Novation te vinden is er in de oriëntatiefase van het project in samenwerking met enkel stakeholders binnen het bedrijf een eisenpakket opgesteld in de vorm van interne en externe kwaliteitsattributen. Aan de hand van deze eisen is er een literatuuronderzoek naar verschillende databases uitgevoerd waarbij is gekeken naar de mogelijkheden die de verschillende systemen bieden en welke hierbij het beste aansluit op de gestelde eisen. Hierbij wordt onderscheid gemaakt tussen de zaken waaraan de database moet voldoen om overwogen te worden en de zaken die als pré gelden. Het document met de volledige afweging is bijgevoegd als bijlage D.

4.1 Voorselectie

De door het bedrijf benoemde primaire kwaliteitsattributen van een geschikt systeem zijn onder te verdelen in de categorieën functionaliteit, betrouwbaarheid en onderhoudbaarheid. Hierbij is er vervolgens als secundaire punten tevens bekeken naar de bruikbaarheid, vervangbaarheid en efficiëntie van de systemen.

Vervolgens is er aan de hand van de primaire kwaliteitsattributen een voorselectie gemaakt van verschillende NoSQL-databases welke voldoen aan dit eisenpakket. Deze voorselectie heeft een set van vier databasesystemen opgeleverd welke vervolgens nader onderzocht zijn: HBase, Cassandra, MongoDB en CouchDB..

4.2 Uitgebreide analyse

De vier overgebleven systemen zijn na de voorselectie uitgebreid getoetst aan de opgestelde lijst van kwaliteitsattributen. Hierbij wordt in het oordeel onderscheid gemaakt tussen de primaire en secundaire kwaliteitsattributen, de primaire kwaliteitsattributen wegen zwaarder voor het bedrijf en deze punten hebben dan ook een dubbele weging gekregen ten opzichte van de secundaire attributen. Wanneer we alle beoordelingen op rij zetten, krijgen we het volgende overzicht:

	MongoDB	CouchDB	HBase	Cassandra
Primaire kwaliteitseisen				
Functionaliteit:	++	+	+	+
Betrouwbaarheid:	++	+	++	++
Onderhoudbaarheid:	+ / ++	+ / -	+	++ / +
Secundaire kwaliteitseisen				
Bruikbaarheid:	++	++ / +	+	+
Vervangbaarheid:	-	-	-	-
Efficiëntie:	+	+	+	+
Eindoordeel:	++ / +	+	+	+

Tabel 1: Scores verschillende NoSQL databases zoals uiteengezet in bijlage D, Research Document NoSQL databases.

In de tabel is zichtbaar dat de databases in het algemeen dicht bij elkaar liggen qua beoordeling. Dat is ook deels logisch door de voorselectie aan het begin van het proces. Daardoor zijn de opties die lager zouden scoren niet meegenomen in de beoordeling. In de eindbeoordeling steekt MongoDB net iets boven de andere drie kandidaten uit, dit systeem scoort op alle fronten goed. Cassandra en HBase scoren vergelijkbaar goed en worden gevolgd door CouchDB waarbij het verschil in de uiteindelijke scores op kleine details aankomt.

4.3 Conclusie

Onze conclusie luidt dat MongoDB het beste scoort bij de opgestelde lijst van kwaliteits- attributen. Op basis van deze informatie lijkt dit de meest interessant optie om toe te passen binnen het bedrijf. MongoDB scoort op alle punten gelijk of beter dan de concurrentie in de test, hoewel de afstand tot Cassandra slechts beperkt is.

In de door E.Novation geschetste use cases is het niet waarschijnlijk dat er binnen afzienbare tijd gewerkt zal gaan worden met datasets van dusdanig grote volumes dat er daadwerkelijk geprofiteerd kan worden van de capaciteit die Cassandra in dit geval biedt. Wanneer dit in acht genomen wordt biedt MongoDB een eenvoudigere infrastructuur, toegankelijker documentatie en gelijkwaardige mogelijkheden dan Cassandra waarmee MongoDB ietwat positiever uit de vergelijking komt. Deze database zal dan ook worden gebruikt in het vervolg van het project.

5 Voor- en nadelen gebruik NoSQL

Onderdeel van het project is het beschrijven van de voor- en nadelen van het gebruik van een NoSQL databasesysteem, specifiek MongoDB, in combinatie met de CQRS en Event Sourcing patterns. Doordat er met het gebruik van CQRS de mogelijkheid ontstaat om voor het opvragen van data en het verwerken van aanpassingen verschillende databasesystemen te gebruiken komen er ook verschillende eisen voor beide systemen naar voren.

Om hier op in te spelen zal eerst gekeken worden naar het gebruik van MongoDB, of vergelijkbare oplossing, als event store om vervolgens een blik te werpen op de implicaties van het gebruik bij een view database. Tot slot zal er ingegaan worden op de applicaties die al dan niet geschikt zijn voor deze combinatie en de losse delen aan zich. Voor uitgebreidere uitwerking van dit alles verwijzen we naar bijlage F: Gebruik Axon en MongoDB.

5.1 MongoDB als event store

In de huidige stabiele versie van het Axon Framework is ondersteuning voor het gebruik van MongoDB als opslagmedium, vergelijkbaar met de ondersteuning voor relationele databases. Hoewel deze implementatie functioneel is en zonder problemen werkt maakt deze implementatie geen gebruik van de specifieke mogelijkheden die MongoDB biedt ten opzichte van relationele databases. De verwachting is hierbij dan ook dat de performance voor deze toepassing in lijn zal zijn met de performance van een relationele database.

De aanpak van het opslaan van data in MongoDB is momenteel identiek aan de methode gehanteerd voor de opslagmethoden in een relationele database en biedt hierdoor ruimte voor verbetering. Zo zou het mogelijk zijn om de events per domeinobject op te slaan in één document. Door deze aanpassing kunnen alle events van een domeinobject op deze wijze opgehaald worden aan de hand van slechts één key. Hierdoor hoeven er geen zoekacties meer uitgevoerd te worden wat de , wanneer opgeslagen aan de hand van de identifier van het domeinobject, met slechts één van één domeinobject met slechts één key opgehaald kunnen worden, waardoor er geen zoekacties uitgevoerd hoeven te worden wat voor snelheidswinst zou kunnen zorgen.

Bij deze aanpak komt wel een limiterende factor aan het licht; één document binnen MongoDB kan momenteel maximaal tot een grootte van 16 megabyte groeien. Wanneer we hierbij rekenen met event-objecten die in XML-vorm opgeslagen worden dan zullen deze events mogelijk gemiddeld 2 kilobyte ruimte per stuk innemen³. Hiermee biedt één document ruimte aan iets minder dan 8192 events. Mochten domeinobjecten in het systeem meer dan dit aantal events kunnen bevatten dan valt deze werkwijze af of moet er een mogelijkheid ingebouwd worden om zeer grote collecties events op te splitsen in meer dan een document waarmee de voordelen af kunnen nemen.

³ Ter referentie: het grootste event in de door ons ontwikkelde applicatie is, naar XML geserializeerd, rond de 1 kilobyte groot. Dit event bevat meerdere attributen met bijbehorende waarden.

Een andere klein voordeel hierbij is dat er gewerkt kan worden met slechts één identifier per domeinmodel, namelijk het id (de Aggregateldentifier) van dit model zonder dat er zowel met een 'id' als een 'Aggregateldentifier' gewerkt wordt voor de events. Hiermee zal de benodigde ruimte voor de index op de collectie afnemen.

Momenteel wordt door het Axon Framework enkel gebruik gemaakt van XML om events naar te converteren voor opslag. Wanneer hierbij in plaats van XML gebruik gemaakt wordt van BSON zal MongoDB deze data efficiënter op kunnen slaan en wordt de serializatie efficiënter. Zo blijft er van een event dat bij XML-serializatie nog 940 bytes inneemt slechts 336 bytes over wanneer hetzelfde object in de vorm van BSON binnen MongoDB wordt opgeslagen. Hiermee kunnen er op deze wijze significant meer events opgeslagen worden in één document en neemt het algehele geheugengebruik van de database af. Bovendien biedt dit de mogelijkheid om tevens voor analyse- of beheersdoeleinden de in het systeem aanwezige events te doorzoeken doordat MongoDB de structuur van de events op deze wijze kan herkennen.

Op dit moment zijn er geen significante voordelen van het gebruik van MongoDB als Event Store, tegelijk geeft het gebruik ook geen specifieke nadelen. Wanneer de punten die hierboven besproken zijn doorgevoerd worden zullen er enkele voordelen ten opzichte van een RDBMS kunnen ontstaan op het gebied van performance, het testen hiervan valt echter buiten de scope van ons onderzoek..

5.2 MongoDB als query-database

Indien MongoDB als view database fungeert biedt de database vooral voordeel als gevolg van het feit dat MongoDB schemaless is en dus geen vast databaseschema in strikte vorm hanteert. Dit heeft als voordeel dat de opslag van data één op één overeen met het gebruikte datamodel kan komen. Wanneer er hiermee vervolgens data aan de gebruiker getoond moet worden kan er gewerkt worden met eenvoudige queries zonder de noodzaak om verschillende tabellen te combineren met JOIN-statements. Dit kan de ontwikkeling vereenvoudigen en de performance verhogen.

Een ander voordeel van de schemaless aanpak is dat de datamigratie in bepaalde gevallen eenvoudiger uit te voeren is. Doordat er geen vast databaseschema aanwezig is kunnen nieuwe velden eenvoudig toegevoegd worden door deze simpelweg naar de database te sturen. Hierover meer in bijlage F.

We kunnen dan ook concluderen dat het gebruik van MongoDB als query-database enige voordelen biedt die de ontwikkeling onder sommige omstandigheden zou kunnen vereenvoudigen en tegelijk de performance kan verhogen wanneer er door het gebruik van documenten joins weggenomen kunnen worden.

5.3 Toepassingen met en zonder profijt

Hoewel het gebruik van NoSQL en CQRS met Event Sourcing vele voordelen lijkt te hebben betreft het ook hier geen ultieme oplossing waar elk probleem mee opgelost zou kunnen worden. Om een beter beeld te vormen over wat het wel biedt zullen we dan ook kort de voor- en nadelen van het gebruik van deze technologieën bekijken.

5.3.1 NoSQL

In de eerste plaats kent het gebruik van NoSQL in het algemeen verschillende voordelen. Vooral wanneer er gewerkt moet worden met zeer grote datasets verspreid over meerdere machines of caching-lagen. Dit zijn punten waarop een RDBMS minder flexibel is, deze hanteren een meer generieke aanpak. Daarbij bieden ze de programmeur veel garanties op het gebied van consistency en transacties (ACID en BASE, bijlage B).

Applicaties die sterk leunen op de garanties omtrent de consistentie van data zullen in de meeste gevallen dan ook beter af zijn bij een RDBMS. Ook applicaties die een sterk relationeel karakter of grote hoeveelheid relationele data bevatten, zullen het mogelijk beter af zijn met een relationele database. NoSQL databases blinken hier minder in uit doordat er geen mogelijkheid tot het uitvoeren van JOIN-queries bestaat en dus, wanneer de data over meerdere collecties verspreid staat, deze data los per collectie opgehaald moet worden wat het opvragen van data minder efficiënt maakt.

In andere situaties kan de keuze voor een NoSQL database die goed aansluit op de specifieke taak voor een hoop extra performance zorgen. Voorbeelden hiervan zijn voorbeeld key-value stores als Memcached⁴, welke met zeer hoge snelheden waarden aan de hand van een specifieke key op kunnen halen. Document stores, zoals MongoDB, zijn bijvoorbeeld weer geschikt om complexe data achter één key op te slaan. Bij een relationele database zou deze data over meerdere tabellen uitgesmeerd moeten worden om vervolgens met behulp van JOIN-queries weer samengevoegd te worden wat een meer kostbare actie zal zijn.

5.3.2 CQRS/ES

Het gebruik van de patterns CQRS en Event Sourcing lijkt daarnaast ook niet in alle gevallen de juiste keuze. De aanwezigheid van een Event Store biedt hierbij zeker voordelen wanneer de applicatie auditing mogelijk moet maken of er op eenvoudige wijze voorgaande versies van een model teruggeplaatst moeten kunnen worden. Het gebruik van de event driven architecture zoals het Axon Framework deze voorschrijft biedt het voordeel van een meer opgesplitste en goed testbare business logic wat de onderhoudbaarheid en het gemak van het doorvoeren van wijzigingen mogelijk ten goede komt.

Wanneer een applicatie echter geen directe meerwaarde hecht aan het gebruik van event sourcing, biedt dit pattern ook geen voordelen en zorgt het eerder voor onnodige complexiteit. In dit geval is het nog wel mogelijk om gebruik te maken van de overige features van de CQRS-implementatie van Axon maar ook dit is niet in alle gevallen de juiste keuze. Voor relatief eenvoudige applicaties met veel CRUD-functionaliteit, een zeer hoge verhouding writes ten opzichte van het aantal reads of near-realtime vereisten biedt deze aanpak weinig voordelen en heeft het mogelijk juist een vertragend en onnodige complicerend effect.

⁴ <http://memcached.org/>

6 Implicaties applicatieontwikkeling

Behalve algemene voor- en nadelen van het gebruik van CQRS/ES en NoSQL zijn er ook implicaties voor het ontwikkelproces en het beheer van applicaties die gebruik maken van deze technologieën welke we in dit hoofdstuk zullen bespreken. Hierbij wordt als eerst gekeken naar de wijzigingen die in het ontwikkelproces doorgevoerd moeten worden, als tweede komen mogelijke wijzigingen aan de te ontwikkelen systemen aan bod waarbij we tot slot ingaan op de implicaties voor het beheer en de mogelijke effecten op efficiëntie van de ontwikkeling.

6.1 Wijzigingen ontwikkelproces

In de vergelijking van de twee verschillende projecten in bijlage E: 'Implementatieanalyse', lijkt de leercurve van het ontwikkelen met CQRS steiler te verlopen dan dat van een meer traditioneel opgebouwd systeem. De implicatie hiervan is dat een project uitgevoerd door programmeurs die minder ervaren zijn er enige tijd uitgetrokken moet worden om bekend te worden met de methodiek en daarmee meer tijd zal kosten.

Ook is het van belang goed te zorgen voor een juiste afstemming van de applicatie die ontwikkeld moet worden met de gekozen patterns en frameworks voordat tot de implementatie overgegaan wordt. De applicatie die wij in ons onderzoek hebben ontwikkeld leende zich niet optimaal voor het gebruik van een event driven architecture zoals het Axon Framework deze kent, waarbij het gebruik hiervan voor een enigszins vertragende factor zorgt en de complexiteit van de architecture onnodig wordt verhoogd.

Pas zodra een applicatie complexere business logic bevat zal de overhead van het gebruik van Axon wegvallen tegen de mogelijk te winnen voordelen van een splitsing van de business logic in meerdere losse onderdelen en bijbehorende overzichtelijkheid en testbaarheid. Het is hierdoor dus zaak een goede afweging te maken voordat men aan de slag gaat met het Axon Framework en CQRS/ES in het algemeen.

Daarnaast blijkt uit bijlage B, 'Oriëntatieverslag', dat een verandering van database met verschillende CAP verhoudingen invloed heeft op of en hoe de applicatie met fouten en out-of-date informatie om moet gaan. Hier is deels rekening mee te houden in de configuratie van de database, zo is MongoDB bijvoorbeeld sterk consistent wanneer er slechts naar één server gecommuniceerd wordt, in veel andere gevallen zal hier echter een afweging gemaakt moeten worden.

6.2 Systeemwijzigingen

Aan de hand van de cyclomatic complexity van de code kunnen we een verschil in structuur bepalen tussen de twee applicaties. De moeilijkheidsgraad van het later toevoegen van functionaliteit of code te testen hangt samen met de complexiteit van de code. Zo is het bij een applicatie die een hoge complexiteitscore per methode kent waarschijnlijk lastiger om het overzicht te bewaren tussen de mogelijke executiepaden wat het doorvoeren van wijzigingen kan vertragen.

Uit de grafieken van de complexiteitsanalyse in bijlage E blijkt dat de complexiteit van zowel methoden als klassen in beide projecten relatief laag is. Dit laat zien dat er in deze projecten geen zeer complexe

zaken in één enkele methode/klasse worden afgehandeld. Dit ligt ook in de lijn der verwachting gezien de beperkte complexiteit van de applicaties.

Wel blijkt dat het gebruik van CQRS/ES een zekere hoeveelheid code met zich meebrengt die niet direct effect heeft binnen de applicatie maar nodig is om het geheel in goede banen te leiden. In het geval van de door ons ontwikkelde applicatie was deze hoeveelheid code relatief groot. Te verwachten is dat wanneer applicaties groeien dit verschil met andere ontwikkelmethoden minder groot zal zijn doordat de code die nodig is voor de algemene infrastructuur een kleiner onderdeel van het geheel uit zal gaan maken.

Mocht er uiteindelijk naar een applicatie gekeken worden die wel een complexere business logic bevat, dan is het mogelijk dat de voordelen van het gebruik van het CQRS-pattern aan het licht komen. Doordat CQRS een opsplitsing van de verschillende ‘concerns’ binnen de applicatie voorschrijft (de command- en query-kant) brengt dit als voordeel met zich mee dat de complexiteit mogelijkerwijs minder geconcentreerd zal zijn en daarmee een eenvoudiger testbaar en daarmee eenvoudiger aanpasbaar geheel op kan leveren.

Om een feature aan de huidige CQRS applicatie toe te voegen kan er simpelweg een nieuwe koppeling aan de command- en eventbus toegevoegd worden. Door losse koppelingen in het CQRS-pattern hoeft de developer niet heel het systeem te kennen, terwijl dit bij een LA-systeem in grotere mate het geval is. Deze losse koppeling maakt het eenvoudiger om nieuwe functionaliteit toe te voegen en daarnaast ook om deze te testen doordat elk onderdeel op zichzelf een deel van het proces op zich neemt.

6.3 Veranderingen applicatiebeheer

Op het gebied van applicatiebeheer zullen er verschillende veranderingen optreden wanneer er gebruik gemaakt gaat worden van CQRS/ES en MongoDB. De verandering van relationele database naar NoSQL en de omschakeling van een traditionele opslag naar een event store hebben daarvan de grootste impact, het gebruik van het CQRS an sich minder.

In plaats van de reguliere relationele database wordt er nu gebruik gemaakt van MongoDB, een NoSQL database. Niet alleen de structuur van de database is anders, ook gebruikt MongoDB geen SQL meer in de communicatie. Via de ‘Mongo shell’ kan de database wel benaderd worden op vergelijkbare wijze als met een relationele database waarna er queries en updates uitgevoerd kunnen worden. Wel kent MongoDB een minder grote hoeveelheid aan beschikbare tooling dan de ‘oude kolossen’ zoals MySQL, SQL Server en PostgreSQL wat waarschijnlijk een omschakeling op het gebied van beheer en de monitoring vereist.

Een wijziging die tevens enige impact zal hebben is de keuze om gebruik te maken van een event store in plaats van een ‘reguliere’ opslag van de verschillende objecten. Door deze omschakeling is het niet meer mogelijk om handmatig rows in de database te wijzigen omdat er op die manier een discrepantie ontstaat tussen de event store en de query-database. Mochten er dan ook fouten optreden in het systeem dan zal er een nieuw event gemaakt moeten worden om deze fouten te corrigeren wat ook een vertraging met zich mee zal brengen.

Tot slot zal de keuze voor het gebruik van CQRS in het beheer weinig veranderingen teweeg brengen. Axon heeft echter wel twee zaken waar in ieder geval rekening mee gehouden moet worden. Zo wordt er standaard met bepaalde fouten, zoals het missen van een event listener voor een bepaald event of het optreden van een exceptie in een event listener, omgegaan door niets te doen. Dit maakt het standaard onmogelijk om op deze fouten op te treden terwijl een dergelijke fout als consequentie kan hebben dat de query-database niet wordt bijgewerkt met de wijzigingen. Deze incorrecte data zou schadelijk zou kunnen zijn voor het gebruik van de applicatie. Voor verdere informatie verwijzen we naar bijlage F en de documentatie van het Axon Framework.

Concluderend kunnen we stellen dat, afhankelijk van de precieze keuzes, er enige implicaties voor het applicatiebeheer zullen zijn. Deze punten zijn al met al niet onoverkomelijk. Wanneer er voldoende rekening wordt gehouden met de valkuilen, er tijd beschikbaar is om de juiste tooling op te sporen en beheerders in te laten werken zou de adaptatie succesvol moeten verlopen.

6.4 Efficiëntie ontwikkeling

Tijdens het project zijn twee featuresets geïmplementeerd, case 1: “toevoegen van zoekfunctie voor contactpersonen” en case 2: “een wijziging in het domeinmodel, een contact kan met attribuut telefoonnummers nu meerdere nummers hebben”. Hierbij is door het uitvoeren van de ontwikkeling ervaring opgedaan en is deze gedocumenteerd. Door de beperkte hoeveelheid datapunten en een focus op het slechts het op doen van ervaring kunnen er geen sterke conclusies aan de resultaten verbonden worden.

Wel hebben we een vergelijking van onze bevindingen gemaakt met een onderzoek uitgevoerd in opdracht van Sogyo dat tevens een vergelijking trekt tussen een standaardmethode van ontwikkelen versus het gebruik van een event driven architecture. Omdat het gebruik van Axon tevens een event driven architecture met zich meebrengt en er in het onderzoek voor Sogyo vergelijkbare resultaten geboekt worden geeft dit een indicatie dat onze resultaten in de juiste richting wijzen. Uitgebreide analyse van de implementatie en referenties zijn te vinden in bijlage E ‘Implementatieanalyse’.

Ondanks de ontbrekende statistische significantie zijn er wel empirische resultaten verkregen en is er veel ervaring opgedaan met beide systemen. Hierbij heeft het werk bij case 1 in ons geval een steilere leercurve opgeleverd bij het bekend worden met het CQRS-project dan de Layered Architecture-tegenhanger. Uit case 2 is gebleken dat voor bepaalde wijzigingen er bij een CQRS-systeem minder systeemkennis nodig lijkt te zijn om het geheel af te ronden wanneer de uit te voeren wijzigingen aansluiten op de command-driven werkwijze. Uit beide cases is gebleken dat de benodigde implementatietijd nagenoeg gelijk is voor aanpassingen aan beide systemen en er met name een verschil in de initiële leercurve lijkt te zitten.

Om te kunnen bepalen waar deze trend uiteindelijk precies op zal wijzen zal kunnen blijken uit een grootschaliger project. Met een groter aantal losstaande wijzigingen en een grotere groep proefpersonen zijn er meer datapunten te genereren, waarna een conclusie volgt die statistisch beter onderbouwd is.

7 Performance MongoDB

Het uitvoeren van performance testing op de uiteindelijk gekozen NoSQL database en ontwikkelde proof of concept-applicaties stond oorspronkelijk op de planning. Het plan was om zo te pogen verschillen in performance te meten tussen beide oplossingen en te kijken of hier significante verschillen te ontdekken zijn. Tijdens het project is dit punt meer naar de achtergrond verschoven nadat bleek dat de performance voor de meeste doeleinden ruim voldoende bleek te zijn en voor E.Novation de prioriteit bij het uitzoeken van andere zaken lag. In overleg met beide begeleiders hebben we besloten hier dan ook geen tijd aan te besteden maar dit punt voor een volgend onderzoek aan te raden.

Om toch enigszins in te gaan op de performance van de NoSQL database, hebben we de paper “Can the elephants handle the NoSQL onslaught?”⁵ bestudeerd. Deze paper maakt een vergelijking tussen de performance van SQL Server en MongoDB. In deze paper wordt een uitgebreide performancetest uitgevoerd met datasets van verschillende groottes op een cluster van meerdere servers. In dit onderzoek spreken de resultaten in het voordeel van SQL Server, met name bij grotere datasets. Wel blijkt ook in dit onderzoek dat MongoDB, met name bij beperktere datasets, goed mee lijkt te kunnen komen met SQL Server. Dit onderstreept ons vermoeden dat de performance voor applicaties die niet onder enorme druk draaien voldoende hoog zal zijn.

8 Proces

Naast de resultaten van het onderzoek is het ook van belang stil te staan bij de procesgang tijdens het project. In dit hoofdstuk zal dan ook gekeken worden naar de initiële planning, gevolgd door de uiteindelijke indeling van de uitgevoerde sprints. Hierbij zullen ook eventuele knelpunten bij de verschillende deliverables kort besproken worden. Vervolgens zal de onderlinge samenwerking, de begeleiding en de ervaringen op het bedrijf besproken worden.

8.1 Initiële planning

In de planning hebben we rekening gehouden met het gebruik van Scrum tijdens het project. Scrum is een proces waarbij elke sprint bepaalde doelen worden gekozen waarbij dus niet vooraf aan volledig plan voor langere tijd uitgestippeld staat maar dit meer afhangt van de prioriteit van de klant of in ons geval de begeleider. De initiële planning is opgenomen als bijlage in het Plan van Aanpak welke te in dit verslag is opgenomen als bijlage C.

8.2 Uiteindelijke projectplanning

8.2.1 Sprint-indeling

Door het gebruik van Scrum is niet vooraf exact vastgelegd wat er tot in de puntjes uitgevoerd zou gaan worden tijdens het project. Deze flexibiliteit heeft aangetoond prettig te kunnen werken, zo is de focus tijdens het project enigszins verschoven van puur NoSQL en bijbehorende performance naar meer een

⁵ <http://pages.cs.wisc.edu/~floratou/SQLvsNoSQL.pdf>

combinatie van NoSQL en CQRS/ES. Wanneer we met een watervalmodel gewerkt zouden hebben dan was zo'n verschuiving niet direct mogelijk geweest door de algehele planning.

De sprints zijn uitgevoerd door per sprint in samenwerking met de begeleider, Bastiaan Bakker, vast te stellen waar de focus op zou moeten liggen om vervolgens aan de slag te gaan met de specifieke invulling hiervan. Uiteindelijk is hier de sprint-indeling zoals uiteengezet in tabel 2 uit voortgekomen.

Taak \ Sprint	1	2	3	4	5	6	7
Plan van Aanpak							
Oriëntatieverslag							
Onderzoek NoSQL							
Opbouw PoC-applicaties							
Implementeren features							
Documentatie implementatie							
Eindverslag							
Interne wiki bijwerken							
Eindpresentatie							

Tabel 2: Verloop werkzaamheden sprints.

Elk blokje in tabel 2 representeert hierbij een sprint, welke in ons geval twee weken van drie (sprint één en twee) of vier (overige sprints) werkdagen omvatte. De indeling in de tabel is niet helemaal strikt, er is enige overlap tussen de perioden die 'hard' door een sprint worden gescheiden doordat er zo nu en dan enige vertraging optreedt bij het afronden van verschillende zaken, mede door de beperkte ervaring met het inschatten van tijd voor dergelijke werkzaamheden.

Allereerst zijn we begonnen met het opleveren van het Oriëntatieverslag en het Plan van Aanpak, respectievelijk in bijlage B en bijlage C, waarbij er enige uitloop is geweest in de tweede sprint. Nadat deze documenten opgeleverd waren zijn we verder gegaan met het onderzoeken van de verschillende NoSQL databases, het resultaat hiervan is bijgevoegd als bijlage D. Na deze fase van documentatie zijn we begonnen aan de twee voorbeeldprojecten welke gebruikt zouden worden voor het uitvoeren van het vergelijkende onderzoek waarna we in sprint 4 en 5 de gewenste features geïmplementeerd hebben.

Nadat dit gedaan was hebben we de bevindingen gedocumenteerd en het literatuuronderzoek naar de nog onderbrekende onderzoeksvragen uitgevoerd, gecombineerd met het opleveren van de conceptversie van het eindverslag in sprint 6 en 7. Tot slot is in sprint 7 de interne wiki bijgewerkt met de conclusies van het bachelorproject en is de eindpresentatie vormgegeven en binnen het bedrijf gepresenteerd.

8.2.2 Oriëntatieverslag en Plan van Aanpak

We zijn gestart met het maken van de door de TU Delft vereiste deliverables, namelijk het Oriëntatieverslag en het Plan van Aanpak. Tijdens deze fase hebben we onderzoek gedaan naar de mogelijke onderzoeksrichting en het opstellen van onderzoeksvragen en om een zo relevant mogelijk onderzoek uit te kunnen voeren voor E.Novation. De onderwerpen die daarin naar voren zouden komen zijn tijdens de gesprekken voor het project vastgesteld waarbij de nadruk is komen te liggen op NoSQL

en het CQRS-pattern. De betreffende onderwerpen hebben we ook uitgediept ter analyse van de probleemstelling in ons Oriëntatieverslag. Vervolgens hebben we de onderzoeksvragen opgesteld, die zijn in overeenstemming met de beide begeleiders goedgekeurd.

Nadat de vragen bekend waren hebben we het Plan van Aanpak geschreven. Daarin staat beschreven wat het project precies inhoud en welke technieken en tooling er gebruikt gaat worden. Tevens staan daar toelichtingen over opdrachtgever en methodieken die we daar gebruiken.

8.2.3 Database-onderzoek

Het logisch vervolg in dit project is het uitkiezen van een NoSQL database die we in de proefprojecten konden toepassen. Eerst hebben we uitgezocht welke databases er allemaal beschikbaar zijn om vervolgens een voorselectie te maken op basis van de eerder vastgestelde kwaliteitsattributen.

Deze voorselectie leverde vier verschillende databases op die in de buurt van de doelstelling kwamen. Deze databases hebben we uitgebreider onderzocht zoals te lezen is in bijlage D. Aan het eind van deze periode hebben we een document op kunnen leveren wat een onderbouwde keuze maakt voor één database die in het vervolg van het project gebruikt gaat worden.

Het lastige in dit onderzoek is geweest om de score van verschillende databases vast te stellen op basis van de kwaliteitsattributen. Niet voor alle databases is uitgebreid informatie over betreffende onderwerpen te vinden en bovendien zijn alle vier de kandidaten nog sterk in ontwikkeling waardoor er een ruime hoeveelheid aan outdated informatie beschikbaar is.

8.2.4 Projecten uniformeren

Nadat bekend is geworden welke database we gaan gebruiken voor de testprojecten, zijn we de testprojecten gaan voorbereiden. Dit is ook de eerste Sprint waarin we serieus met Scrum aan de slag zijn gegaan. We hebben de zogenaamde Backlog gevuld met taken die in deze sprint moeten gebeuren. Bij het vullen van de backlog moet er ook een tijdschatting gedaan worden om te kunnen bepalen wat er in twee weken gedaan kan worden. Dit was een lastig punt, vooral omdat wij nog weinig ervaring met het ontwikkelen op deze manier hebben. We hebben toen een grove schatting gemaakt van tijd die het zou kosten om alle taken uit te voeren.

Het doel van deze sprint was om de twee verschillende projecten zou gelijk mogelijk te maken. De backlog is gevuld met allerlei taken die aan dit doel bijdragen. De beide projecten maken nu gebruik van Spring MVC en hebben dezelfde frontend, het verschil zit in het pattern en in de gebruikte database. Tijdens deze fase is onze begeleider enkele weken weg geweest en kregen we ondersteuning van Patrick Maat, die is opgetreden als waardige vervanger.

Aan het eind van de sprint hadden beide projecten op alle fronten gelijkwaardig moeten zijn zodat we vervolgens zo zuiver mogelijk aan de slag zouden kunnen met de implementatie. Echter is dat niet allemaal in één sprint gebeurd. De tijdschatting is niet helemaal nauwkeurig geweest, daardoor hebben we wat taken door moeten schuiven naar de volgende sprint. Volgens de vervangende begeleider is dit niet erg, aangezien we dit door kunnen schuiven. Wel moeten we opletten dat we niet alles zomaar

doorschuiven, dan krijgen we op het einde een ophoping van taken die nog niet zijn uitgevoerd wat uiteraard niet de bedoeling is.

8.2.5 Projecten uitbreiden

Nu de projecten in de gewenste staat verkeren, kunnen ze worden uitgebreid. De uitbreiding staat in het teken van de vergelijking tussen de twee verschillende projecten waarbij de benodigde tijd om een aanpassing door te voeren werd gemeten.

Een nieuwe sprint houdt in dat er nieuwe schattingen voor de tijd gedaan moeten worden. Aan het einde van deze sprint kwamen de tijden al beter overeen met de planning. Ook hier zijn we weer enigszins uitgelopen, vooral de uitbreiding hebben meer tijd gekost dan ingecalculeerd.

Tijdens de fase waarin we de projecten hebben uitgebreid hadden we een paar punten waarover we hebben getwijfeld of niet helemaal uit zijn gekomen. Op deze momenten konden we op de ondersteuning van de collega's op dezelfde kamer rekenen. Voorbeelden van problemen zijn het debuggen van Spring's Autowiring en de meest 'pure' aanpak om bepaalde functionaliteit te verwerken in CQRS.

Na deze sprint hebben we een Sprint demonstratie gehouden. In deze demo hebben we de functies en taken die we hebben uitgevoerd gedemonstreerd. Hierbij waren verschillende collega's aanwezig om op te hoogte te geraken van de ontwikkelingen tijdens dit project. Tijdens deze demo hebben we veel nuttige feedback gehad om te verwerken in het project. Zo gaf de naamgeving van het project enige verwarring bij de collega's, intern is het project namelijk aangeduid als '(Onderzoeksproject) NoSQL' terwijl de focus uiteindelijk meer op het toepassen van CQRS/ES met MongoDB als opslagsysteem heeft gelegen. We hebben er in het vervolg naar gestreefd om dit te nuanceren zodat er minder verwarring ontstaat over de inhoud van het project.

8.2.6 Analyse van verzamelde data

Tijdens de analyse van de data zijn we er achter gekomen dat we sowieso geen statistisch onderbouwde conclusies kunnen geven. Dit heeft te maken met de beperkte set aan data die we hebben verkregen met onze metingen.

Vanuit het oogpunt van E.Novation bleek het namelijk interessanter om meerdere invalshoeken te belichten wat in overleg met de begeleider Jan Hidders een geschikte invalshoek leek te zijn. We hebben ook ondervonden dat de informatie die we voor de analyses nodig hebben gehad, beter hadden kunnen bijwerken tijdens het ontwikkelen zelf. In sommige gevallen hebben we tijd nodig gehad om te reconstrueren wat we hadden gedaan wat zonde van de tijd is.

De analyses geven niet op alle opgestelde hoofd- en deelvragen antwoord. Na deze analyses hebben we een plan geopperd om de performance te testen, dat is immers één van de nog niet behandelde deelvragen. De opdrachtgever van E.Novation gaf hier aan de performance toch niet heel relevant te vinden. Op dat moment hebben we met beide begeleiders gezamenlijk overleg gepleegd om een alternatief plan te formuleren. Daarin hebben we vastgesteld om ons meer te richten op de andere deelvragen en performance marginaal te behandelen.

Bastiaan heeft een alternatief voorstel gedaan om nog te behandelen, de behandeling van de verschillen tussen ACID en BASE databases is relevanter voor E.Novation. Hierbij passen we ons aan op de wensen van de klant, die een nuttige aanvulling op ons onderzoek vormt.

Na de analyses zijn we verder gegaan met onderzoek over ACID/BASE en de zwakke punten van CQRS/ES (Axon) en MongoDB. Deze onderwerpen zijn interessant voor het bedrijf en geven antwoord op enkele van de nog open deelvragen.

Over het algemeen kunnen we stellen dat het proces vrij soepel is verlopen, er zijn geen grote tegenslagen geweest. We hebben wat moeten wennen aan Scrum en de situatie binnen het bedrijf. We zijn er met de intentie ingestapt om veel uit te zoeken over NoSQL, uiteindelijk is de focus enigszins verlegt naar NoSQL en CQRS/ES, dit is conform de wensen van de klant..

8.2.7 Documentatie

Ter interne documentatie is de code is geannoteerd met JavaDoc. Hierbij is bij iedere klasse en iedere niet-triviale methode beschreven wat het doel en mogelijke side-effects van de betreffende code zijn. Dit is tijdens het ontwikkelen van de code geüpdatet met het maken van de methodes en klassen en richting het eind van het project nogmaals herzien om een consistent geheel op te leveren.

Op de Wiki van E.Novation staat de essentiële informatie voor het draaien van de code en begrijpen van de projecten. Er staat onder andere theoretische informatie op over MongoDB en het CQRS pattern en Event Sourcing. Dit wordt in de laatste dagen nog aangevuld met de laatste bevindingen, zodat de ontwikkelaars van E.Novation volledig op de hoogte raken van alle ins en outs van het project.

8.3 Samenwerking

Bij samenwerking is goede communicatie belangrijk, tijdens dit project hebben we eigenlijk altijd in elkaars bereik gezeten tijdens het werken wat overleg eenvoudig heeft gemaakt. De beperkte tijd dat we niet bij elkaar zaten of tegelijk hebben gewerkt, planden we zo dat er veel zelfstandig werk gedaan kon worden en hebben we gewerkt met overleg via Skype om eventuele zaken af te kunnen stemmen.

De werkverdeling hebben we gedaan door zoveel mogelijk één story/takenpakket per persoon toe te wijzen volgens de Scrum-werkwijze. Dit hielden we bij in VersionOne, een digitaal project planning en issue tracking systeem waarin, na een pittige leercurve, alle Scrum-administratie bijgehouden kan worden. Het handige aan deze werkwijze is dat er geen twee personen tegelijk aan één taak kunnen zitten en dus ook niet door elkaar kunnen werken. Op deze manier ging de verdeling erg soepel.

Voor de synchronisatie van de bestanden hebben we een Git repository, ondergebracht bij GitHub, en Google Docs gebruikt. Beide zijn overal met een internetconnectie te bereiken, dit had als achterliggende gedachte dat we niet gelimiteerd zouden zijn tot het kantoor bij E.Novation om aan de slag te kunnen met het project. Daarnaast is het mogelijk om tegelijk in verschillende branches aan dezelfde projecten te werken zonder dat er vervelende corruptie fouten optreden of zaken door elkaar gaan lopen.

Over het algemeen is de samenwerking dus goed verlopen, tijdens het project hebben we ook nog afspraken gehad die niet relevant zijn voor dit project. Voor beide was het geen probleem om flexibel met deze tijd om te gaan. We hebben ook thuis kunnen werken met de gebruikte tools, daar hebben we in deze situaties veel voordeel gehad. Voor deze situaties en de omgang met de werktijden die we hebben aangehouden zijn we erg flexibel gebleken. We hebben elkaar de ruimte gelaten, wat de samenwerking heeft bevorderd.

8.4 Begeleiding

Voor aanvang van het project zijn er twee begeleiders aangesteld, vanuit de TU Delft en vanuit E.Novation. Deze mensen zijn er om al onze vragen te beantwoorden en ons op het juiste spoor te houden. De begeleider van E.Novation is tevens de personificatie van de opdrachtgever.

8.4.1 Begeleiding van TU

We hebben de begeleiding van Jan Hidders als zeer prettig ervaren. Er heeft twee keer overleg plaatsgevonden, in het begin en op twee derde van het project. In het begin hebben we samen met de begeleider van E.Novation besproken hoe het project ingevuld zou gaan worden. Hierbij hebben we nuttige feedback en suggesties ontvangen om het geheel aan te pakken.

Tijdens het project hebben we tweewekelijks per e-mailcontact gehad met Jan Hidders om hem op de hoogte van de voortgang te brengen. Ook hebben we deze momenten gebruikt om Jan vragen te stellen over de vorm of inhoud van het project. Richting het einde van het project hebben we opnieuw een afspraak met beide begeleiders gepland om te bepalen hoe de laatste periode er uit ging zien. Ook in deze fase heeft Jan Hidders ons een nuttig artikel aangeraden wat uiteindelijk ook verwerkt is in het project.

We hebben veel gehad aan de feedback van Jan Hidders, vooral op de punten waarin dit project afwijkt van andere projecten. De aard van het project is onderzoek in opdracht van E.Novation naar NoSQL in combinatie met CQRS/ES, waarbij meer documentatie wordt opgeleverd en een kleine toepassing. We ervaren het als zeer positief dat Hidders zich hard maakt voor de alternatieve aanpak van dit project.

8.4.2 Begeleiding van bedrijf

Vanuit het bedrijf zijn we begeleid door Bastiaan Bakker, met deze begeleider hebben we elke dag een Daily Standup Meeting waarin de voortgang wordt besproken. Er is vanzelfsprekend ook meer contact met Bastiaan geweest dan met Jan Hidders. Bastiaan is tevens opgetreden als opdrachtgever van ons project.

Bastiaan is als teamleider een druk bezet persoon, dat maakte afspraken plannen soms lastig waardoor we snel vaardigheid met het gebruik van Microsoft Outlooks Calendar-functionaliteit opgedaan hebben. Echter is er geen enkel moment geweest waarop er geen mouw aan te passen was. Daarnaast hebben we voor kleinere vragen ook altijd een beroep kunnen doen op de drie collega's die in dezelfde kamer werkten en altijd klaar stonden om ons te adviseren of helpen met bepaalde zaken.

Omdat Bastiaan ook als opdrachtgever fungeert, kan hij de richting van het onderzoek controleren en zo nodig bijsturen. Hij weet ook al het één en ander van de onderwerpen die we behandelen, dat is prettig

zodat hij ook inhoudelijk kan reageren. Daarnaast weet hij goed wat hij wil dat we onderzoeken. We houden een goed gevoel over aan de samenwerking met Bastiaan Bakker.

Tijdens dit project is Bastiaan een aantal weken niet beschikbaar geweest, tijdens deze periode is Bastiaan vervangen door Patrick Maat. Patrick heeft vooral de DSM's geleid wat tevens een prettige ervaring is geweest.

8.5 Ervaringen op het bedrijf

Op het bedrijf zaten we in een kamer met nog drie andere collega's op de afdeling Application Development. Deze afdeling coördineert en ontwikkelt applicaties van E.Novation. Deze afdeling bestaat uit verschillende delen, waar wij vooral te maken hadden met de ontwikkelaars, onze kamergenoten.

8.5.1 Werkplek

Nadat het contract getekend was, zijn we op een korte termijn begonnen. Op dat moment was de workstation waarop wij zouden werken nog niet binnen. We hebben de oriëntatie en het vooronderzoek op twee Windows laptops gedaan aan hetzelfde bureau. Dit was qua ruimte niet optimaal, het was dan ook maar tijdelijk.

Aan het eind van de oriëntatiefase kwamen was het workstation beschikbaar en hebben we beide een aparte werkplek gekregen. Zo hadden we de ruimte, wat efficiënter heeft gewerkt. Het bleek niet mogelijk te zijn om op een praktische manier twee aparte clients te draaien op een enkele workstation. Zodoende hebben we een werkplek gemaakt met een laptop die met behulp van SSH door middel van Putty en Xming een connectie maakt met het workstation om zo toch op Linux te kunnen ontwikkelen en te kunnen profiteren van de vlottere hardware in deze machine.

8.5.2 Collega's

We hebben vooral veel te maken gehad met de collega's die bij ons op de kamer werken. Deze collega's (Floris Korbijn, Jorn Visser en Key Fey Wong) zijn altijd behulpzaam geweest, voor elke vraag konden we bij één van hen terecht. Zo heeft Jorn ons enkele malen geholpen met het oplossen van @Autowiring problemen die Spring veroorzaakt. Floris en Key hebben ons ook enkele malen geholpen met het beantwoorden van vragen. Met de andere collega's hebben we niet veel te maken gehad op het gebied van ons project maar de lunchpauzes hebben de ruimte geboden om voldoende te kunnen praten met enkele collega's.

8.5.3 Ervaringen Scrum

Binnen E.Novation wordt er ontwikkeld volgens het Agile programming principe Scrum. In het Oriëntatieverslag (bijlage B) wordt hier meer uitleg over geven op het gebied van de werkwijze en terminologie. Vorige projecten van afstudeerders met Scrum zijn het bedrijf erg goed bevallen, zodoende zijn wij er ook aan begonnen.

In ons geval is de Product Owner en de Scrum Master één en dezelfde persoon, de owner kan zich nu te veel met de technische invulling van het product bemoeien. Wij hebben dit niet zo ervaren, Bastiaan heeft ons vrijgelaten de meeste dingen zelf uit laten zoeken. In de rol als zodanig heeft Bastiaan zich meer als Product Owner opgesteld.

Toen we bij de eerste sprint van ontwikkeling zijn begonnen hebben we vooral moeite gehad met het inschatten van de tijd die een taak kan gaan kosten. Daardoor liepen de sprints niet helemaal strak bij de gestelde grens van twee weken, maar liepen ze in elkaar over. Dit heeft te maken met de onervarenheid met Scrum en de ontwikkeling van dit soort projecten. Bij de tweede sprint ging het al beter, maar niet significant. We hebben ook niet echt twee keer dezelfde story gedaan, daardoor hebben we ook de opgedane ervaring niet optimaal kunnen benutten. Met meer ervaring zullen we beter in kunnen schatten wat een taak qua tijd kost, daardoor zullen de sprints ook strakker afgebakend zijn.

Er waren ook factoren die het gebruik van Scrum niet bevorderen. Het project hebben we met twee personen uitgevoerd, normaliter is een Scrum team groter. Zeker omdat wij bijna altijd samen aan het werk zijn geweest is een DSM niet altijd nodig. Het werkt ook niet optimaal bij het maken van document, daar is de opdeling toch minder strict dan bij een software ontwerp. Door deze vage grens is het lastig goed gebruik te maken van scrum. Doordat bij deze situatie het anders loopt, hebben we VersionOne, dat is het digitale systeem waarin alle taken worden bijgehouden, ook niet helemaal up-to-date gehouden aan het einde. We hebben de backlog nog apart moeten bijwerken.

Over het hele project gekeken hebben we een goed beeld van wat Scrum inhoud. We hebben het gevoel dat dit systeem bijdraagt aan de efficiëntie van de ontwikkeling. Echter is het niet overal even geschikt voor. Voor de ontwikkeling van een applicatie zouden we het zeker nog een keer gebruiken.

8.6 Conclusie

Na een korte periode van gewenning aan de nieuwe omstandigheden op het bedrijf en ontwikkelomgeving hebben we het project vrij soepel doorlopen. Dat is te danken aan onder meer de prettige begeleiding en de prettige omgang en hulp van de directe collega's. Zoals in alle gevallen is niet alles volledig vlekkeloos verlopen, zo bleek het werken met Scrum bij zaken die niet heel voorspelbaar zijn in sommige gevallen uitdagend en zijn we door te hoge ambitie en een tijdgebrek niet de volledige backlog doorgekomen. Alles in acht genomen kunnen we echter wel terugkijken op een project dat vrij soepel is verlopen.

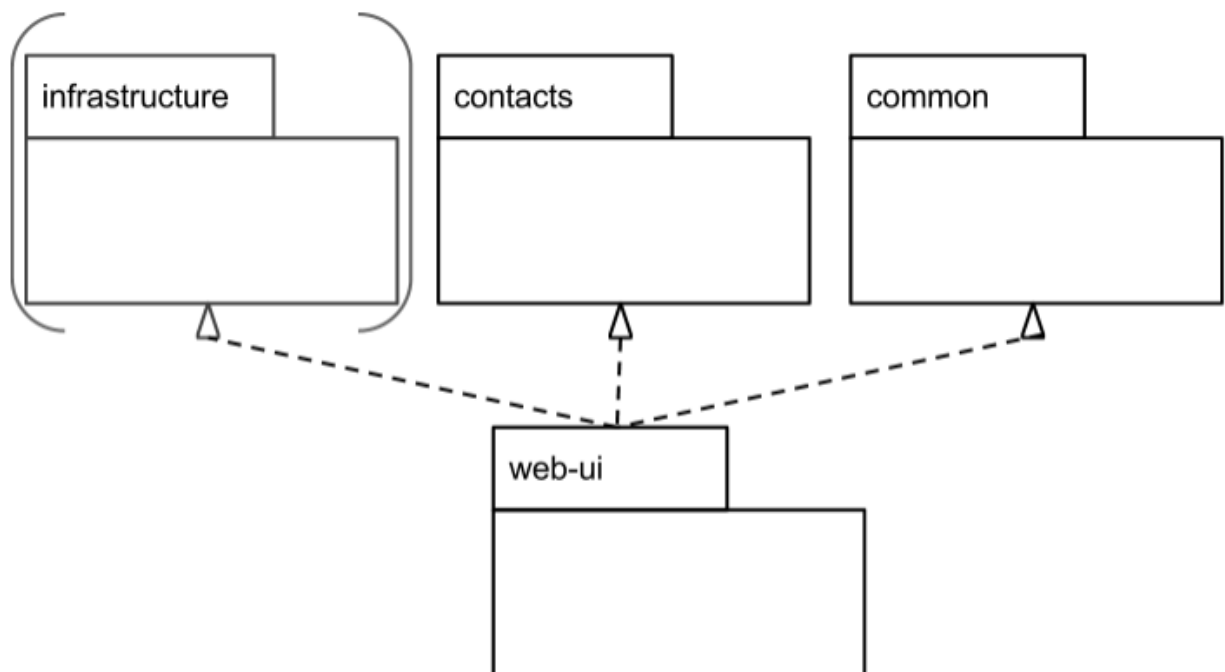
9 Ontwerp

Tijdens het project zijn twee applicaties ontwikkeld met beide een eigen ontwerp om op deze manier inzicht in de verschillen tussen beide methoden te verkennen. De ontwerpwijze van beide applicaties zullen dan los van elkaar besproken worden, waarbij eerst stilgestaan zal worden bij de opbouw van het CQRS-project (Command Query Responsibility Segregation) om vervolgens in te gaan op de Layered Architecture-applicatie.

9.1 Algemeen ontwerp

In grote lijnen bevatten de beide applicaties een soortgelijk ontwerp. In figuur 1 staat dit schematisch weergegeven. Beide projecten bevatten een Maven-project 'contacts' wat de feitelijke kern van de applicatie bevat met de business logic en het gehele domeinmodel. Het Maven-project 'web-ui' bevat vervolgens een Spring MVC-applicatie welke de controllers en views van de applicatie bevat en daarmee een interface naar de gebruiker biedt. Een deel van deze interface is vervolgens afkomstig van het project 'common', waarin onder andere de Java Server Pages zijn ondergebracht om hiermee de code duplication te reduceren.

Tot slot bevat het CQRS-project daarnaast ook nog een Maven-project 'infrastructure', welke enkele hulpklassen biedt en de benodigde configuratie bevat voor het gebruik van MongoDB in combinatie met CQRS.



Figuur 1: Algemene indeling projecten

9.2 Opbouw CQRS

Voordat de opbouw van het CQRS-project beschreven wordt zal eerst ingegaan worden op een korte introductie van het principe van CQRS. Voor een uitgebreidere blik op de werking van CQRS kan in bijlage F het document ‘Analyse toepasbaarheid Axon en MongoDB’ gevonden worden. Dit document bevat een meer gedetailleerde beschrijving waarbij ook ingegaan wordt op de voor- en nadelen van deze aanpak.

9.2.1 Introductie CQRS

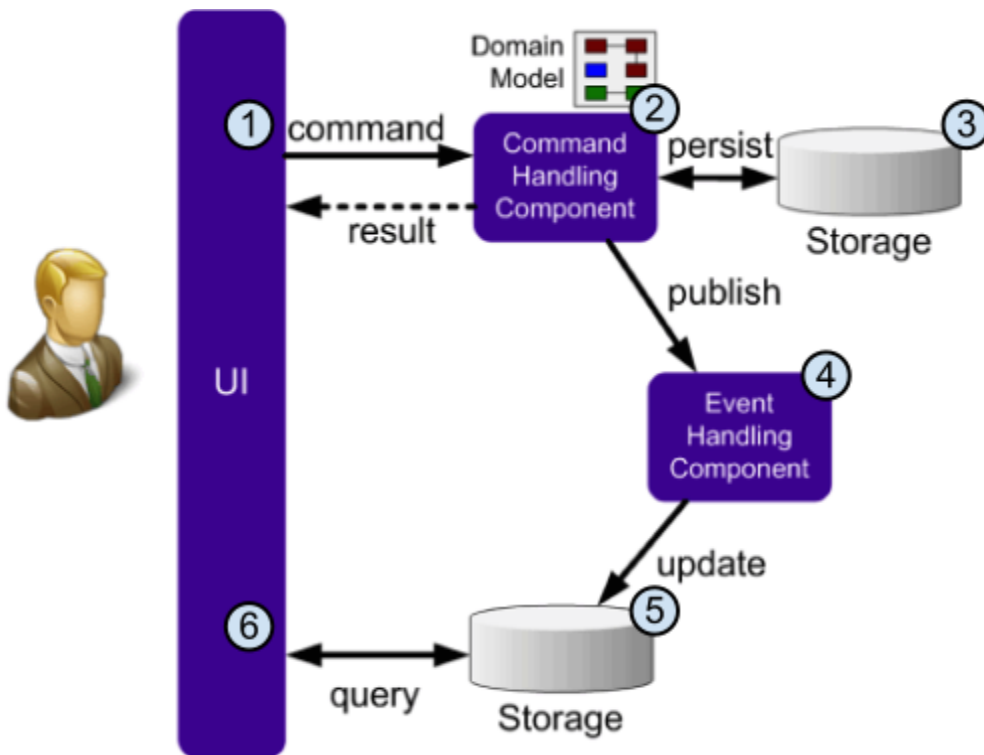
Het ‘Command Query Responsibility Segregation’-pattern schrijft voor dat de verantwoordelijkheid van het uitvoeren van commands en queries gescheiden van elkaar moet worden afgehandeld. Binnen E.Novation wordt een framework gebruikt wat de implementatie van CQRS/ES faciliteert, namelijk het ‘Axon Framework’⁶. Dit framework biedt een infrastructuur om een applicatie op te zetten die het CQRS-pattern volgt. Tevens geeft het handvatten om enkele andere patterns te gebruiken die hiermee goed te combineren zijn.

Axon biedt een combinatie van CQRS, (optioneel) Event Sourcing en een Event Driven Architecture waarmee een applicatie opgezet kan worden. Ook wordt de benodigde infrastructuur meegeleverd om commands en events af te leveren bij de juiste klassen. Met deze infrastructuur kunnen objecten eventueel op ‘event sourced’ wijze opgeslagen worden. Hierbij wordt in plaats van de huidige staat van een domeinobject de reeks aan gebeurtenissen, in de vorm van events, opgeslagen in de database. Door deze events vervolgens nogmaals op te halen kan de uiteindelijke staat van het domeinobject weer teruggehaald worden.

⁶ <http://axonframework.org/>

9.2.2 Systeemontwerp

In een applicatie die opgebouwd wordt met behulp van het Axon Framework zal in de standaardgevallen een architectuur ontstaan die overeenkomt met de schematische weergave van figuur 2.



Figuur 2: Systeemopzet voor een applicatie met het Axon Framework⁷.

Wanneer we de figuur volgens de nummering nalopen geeft dit een beeld van de flow binnen de applicatie. Bij (1), wat een controller van de webapplicatie zal zijn, wordt via interactie met een (user) interface een command aangemaakt. Dit command zal betrekking hebben op het domeinmodel en kan bijvoorbeeld acties omvatten als het aanmaken van een nieuwe contactpersoon. Na het aanmaken van dit command zal deze gepubliceerd worden om (mogelijk asynchroon) door een command handling component opgepakt te worden.

Bij (2) wordt dit command opgepakt en toegepast. Daarbij wordt een event aangemaakt en deze wordt opgeslagen in de command database (3) indien het een event sourcing pattern betreft.

Vervolgens zal het gepubliceerde event door een event handling component (4) opgepakt worden. Aan de hand van het event zullen de opgeslagen entries in de query-database (5) bijgewerkt worden met de

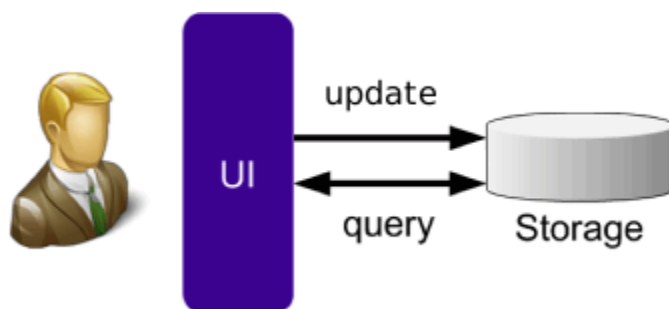
⁷ 'A Discussion with Allard Buijze on CQRS with the Axon framework', J. Long, 24 maart 2010, http://www.infoq.com/articles/cqrs_with_axon_framework

wijzigingen aan het domeinmodel. Wanneer een gebruiker vervolgens de applicatie gebruikt en hierbij data opgevraagd zal worden (6) dan zal hierbij de data uit de query-database tonen.

In de door ons gebouwde applicatie wordt de opslag bij zowel (3) als (5) verzorgd door MongoDB, er wordt met volledig gescheiden collecties binnen de database gewerkt voor de command- en query-zijde van de applicatie.

9.3 Opbouw LA

De opbouw van het Layered Architecture-project is in tegenstelling tot het CQRS-project vrij eenvoudig. De flow door de applicatie is, wanneer we op gelijke wijze naar deze applicatie kijken als eerder gedaan binnen de CQRS-applicatie, weer te geven met onderstaande figuur.



Figuur 3: Systeemopzet voor een standaardapplicatie.

Deze variant van de applicatie maakt geen gebruik van een command bus, de publicatie van events of losse databases voor de queries en commands maar werkt met één en dezelfde backend. Acties zullen in dit geval dan ook uitgevoerd worden aan de hand van gebruikersinteractie welke binnen komt via een van de controllers van de web-ui. Deze controllers zullen vervolgens via de logica welke aanwezig is in de models zelf en de logica beschikbaar via de ContactRepository de gewenste acties uitvoeren.

10 Implementatie

Na de planning van de opbouw van de projecten is overgegaan tot de implementatie van het project. De specifieke implementatie hiervan zal in dit hoofdstuk besproken worden. Hierbij wordt eerst het algemene deel van de implementatie besproken, waarna er ingegaan wordt op de specifieke implementatie van het CQRS en LA-project. Tot slot bespreken we de testmethodiek en het oordeel van de Software Improvement Group over de onderhoudbaarheid van beide projecten.

10.1 Algemene implementatie

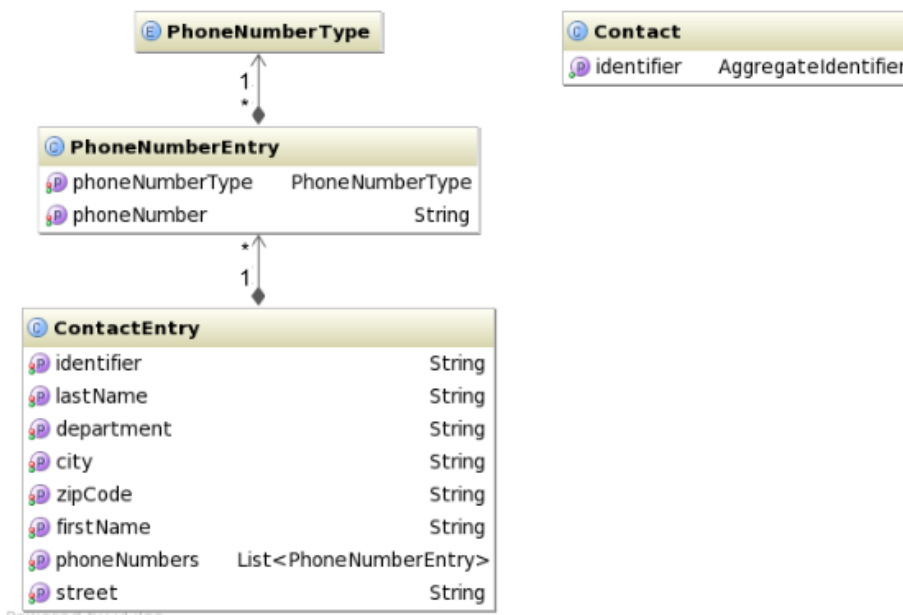
De implementatie van beide applicaties heeft zoals ook besproken in het hoofdstuk 'Ontwerp' geleid tot een web-ui Maven-project met vrij grote overeenkomsten, met name in de aanwezige Java Server Page (JSP) bestanden die de code bevatten die uiteindelijk naar de gebruiker als HTML gerenderd wordt. De afhandeling van de interactie met de gebruiker en de rendering van de templates wordt in beide projecten gedaan met behulp van het Spring MVC framework.

In de laatste fase van het project hebben we er hierbij voor gezorgd dat deze bestanden in een los Maven-project ondergebracht zijn om op die manier te zorgen voor een zo beperkt mogelijke hoeveelheid geduplicateerde broncode om zo de onderhoudbaarheid van de twee applicaties op een zo hoog mogelijk niveau te houden.

10.2 Implementatie CQRS

10.2.1 Domeinmodel

De adresboekapplicatie kent een betrekkelijk eenvoudig domeinmodel wat in onderstaande figuur weergegeven wordt. Hierbij wordt de linker kolom gevuld met het model dat in de query-database opgeslagen wordt en de rechterkolom het model zoals dat bestaat in de command-database.

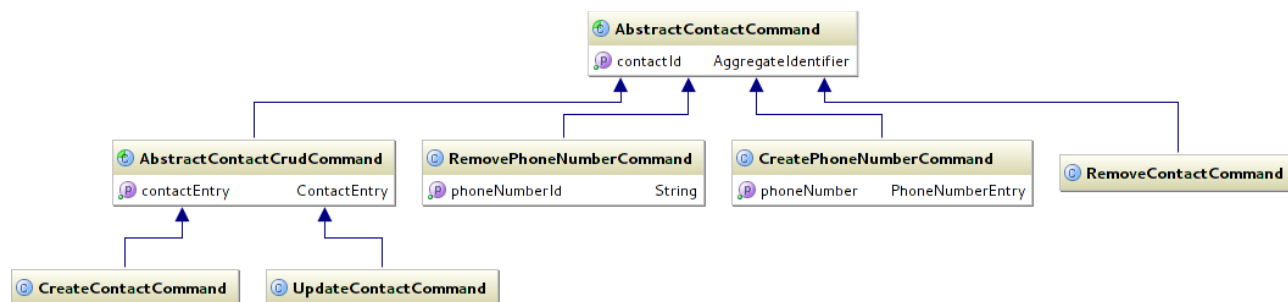


Figuur 4: Domeinmodel van de applicatie.

Zoals in figuur 4 te zien is bevat de applicatie een eenvoudige entry per contact met de nodige properties voor naam, adres en dergelijke. Bovendien bevat de ContactEntry een lijst van PhoneNumberEntry-objecten welke in de database ook op deze wijze embedded in het document van de bijbehorende ContactEntry wordt opgeslagen. Een PhoneNumberEntry bevat vervolgens een property phoneNumber en een PhoneNumberType welke door de gebruiker gebruikt kunnen worden om onderscheid te kunnen maken tussen verschillende soorten telefoonnummers.

10.2.2 Commands

Zoals in voorgaande paragraaf besproken bevat de applicatie een hoeveelheid commands welke de mogelijke acties in het systeem representeren. Deze commands zijn op schematische wijze weergegeven in onderstaande figuur 5.



Figuur 5: Commands in het CQRS-project.

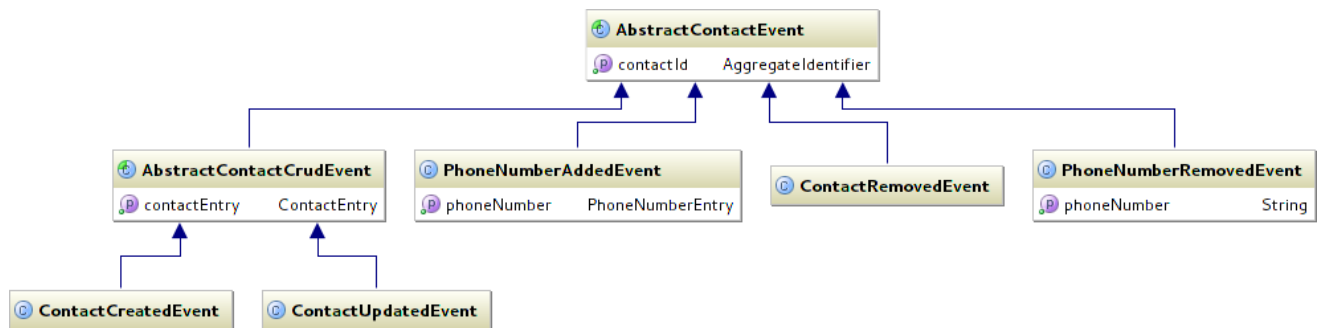
De opbouw bestaat hierbij uit een AbstractContactCommand welke een 'AggregateIdentifier' bevat die tot een unieke Contact in de applicatie behoort en hiermee de basis vormt van de commands in de applicatie. Deze klasse vormt de basis van de aanwezige commands in de applicatie.

De overige commands breiden deze abstracte klasse uit en geven hiermee een specifieke taak weer. Deze klassen bevatten daarnaast ook eventuele data die nodig is bij de uitvoer van het command, zoals bijvoorbeeld een telefoonnummer dat moet worden toegevoegd.

Om de eerder beschreven commands te verwerken bevat het systeem tevens een command handler om deze acties te verwerken. In dit geval is dat een ContactCommandHandler welke per command een losse functie als handler bevat die de interactie met de event store afhandelt.

10.2.3 Events

Naast de hiervoor besproken commands zijn er ook een set events in het systeem aanwezig deze beschrijven de uitvoer van een command. Deze events zijn onderstaand in figuur 6 te schematisch weergegeven.



Figuur 6: Events behorende bij de commands uit figuur 5.

Wanneer deze figuur vergeleken wordt met figuur 4 valt op dat er per command één event aanwezig is. Door de beperkte complexiteit van de logica in de applicatie zijn er momenteel geen scenario's waarin er door één command uit te voeren meerdere events gepubliceerd worden of waarbij er significante verschillen in de beschikbare data in de command- en query-database bestaan.

Vergelijkbaar met de infrastructuur bij de commands bevat het systeem ook een event listener, in dit geval `ContactListener`, welke per event een functie als handler bevat. Deze functies verzorgen vervolgens de interactie met de query-database en zorgen er zo voor dat de gepresenteerde data na verwerking weer overeen komt met de staat van het model in de command database.

10.3 Implementatie LA

Naast de applicatie die gebruik maakt van het CQRS-pattern en is er tevens een applicatie uitgewerkt welke meer op een traditionele wijze zonder het gebruik van events en dergelijken.

10.3.1 Domeinmodel

Het domeinmodel binnen de Layered Architecture-applicatie is op enkele kleine wijzigingen na identiek aan het model uiteengezet voor de query-zijde van CQRS-applicatie op het gebied van de aanwezige properties en relaties tussen de modellen. Voor een overzicht van de modellen verwijzen we daarom de lezer graag naar figuur 4, hierbij moet wel in acht genomen worden dat het LA-project enkel de klassen in de linker kolom bevat, er is hier namelijk geen sprake van een verschil in model tussen de query- en commandzijde van de applicatie.

Vanzelfsprekend houdt de gelijkenis op na de aanwezige properties en relaties, zo wordt er binnen de LA-applicatie voor de opslag gebruik gemaakt van Hibernate. Ook worden, door het gebruik van een relationele database, telefoonnummers niet binnen een contact opgeslagen maar in een eigen tabel met foreign key die wijst naar het bijbehorende contact.

10.4 Testen

Beide applicaties zijn voorzien van unit tests met behulp van het bekende JUnit⁸ test-framework. Om niet voor elke test case veel overbodige zaken op te hoeven bouwen in de gewenste toestand voordat deze getest kan worden hebben we hierbij tevens gewerkt met de ‘mocking’ library Mockito⁹. Met deze library kan eenvoudig een ‘mock’ gemaakt kan worden van een bestaande klasse welke op voor gedefinieerde wijze reageert op functie-aanroepen of kan valideren of een bepaalde methode op de juiste wijze wordt aangeroepen.

Het gebruik van Mockito komt bijvoorbeeld van pas bij het testen van de controllers van de webinterface, hierbij kan er door het gebruik van een mock bij een validatiestap meteen getest worden of de controller de juiste acties onderneemt bij het mislukken van validatie van de gebruikersinput. Wanneer hier zonder een mock te gebruiken getest wordt dan moet er door de programmeur eerst gezorgd worden voor een model wat wordt gevalideerd voordat dit gedrag getest kan worden, waardoor je in de praktijk bezig bent met zaken die je niet op die locatie wilt testen.

Naast het schrijven van unit tests naar eigen inzicht hebben we ook gebruik gemaakt van Sonar, een applicatie die binnen E.Novation toegepast wordt om continu de kwaliteit van code te monitoren. Hierbij hebben we gestreefd naar een voldoende hoge code coverage van de business logic binnen de applicatie en daarbij een zo hoog mogelijke compliance op het gebied van de code guidelines die hierbij staan ingesteld.

10.5 Kwaliteitsoordeel SIG

Nadat we, zoals in hoofdstuk 4 ‘Onderzoek NoSQL Databases’ beschreven hebben, aan de hand van de kwaliteitsattributen als analyseerbaarheid, aanpasbaarheid en testbaarheid een analyse hebben gemaakt is de code ook naar de Software Improvement Group (SIG) gestuurd om een van hun medewerkers een oordeel te laten vellen.

We hebben hierbij telefonisch contact gehad met SIG om te bekijken of het mogelijk zou zijn om de twee simpele applicaties door te lichten en tevens te bekijken of er een verschil in onderhoudbaarheid ontdekt kan worden. Ons verzoek bleek een unieke vraag te zijn in de context van een bachelorproject maar het bleek geen probleem te zijn om tevens een vergelijking tussen de twee projecten te maken.

Het oordeel van de SIG is uiteindelijk neer gekomen op een score van 4 sterren (uit 5) wat betekent dat de code bovengemiddeld onderhoudbaar is, dit betreft zowel de gecombineerde als de individuele code van de twee projecten. De hoogste score is uiteindelijk niet bereikt doordat beide applicaties relatief veel lange code in de JSP bestanden hebben zijn. Voor de twee applicaties gecombineerd is deze score niet bereikt doordat de JSP files van beide projecten een hoge dupliciteit kennen, de views van beide projecten zijn namelijk in de eerste versie nagenoeg gelijk.

⁸ <http://www.junit.org/>

⁹ <http://code.google.com/p/mockito/>

De aanbeveling die wij hierbij hebben gekregen is om te zorgen voor een gemeenschappelijke interface om op die manier dit aspect uit te vergelijking weg te nemen. We zullen dit punt zover mogelijk mee proberen te nemen in de planning van de laatste fase van het project om eventueel toch de 5 sterren-beoordeling te kunnen benaderen.

11 Conclusie

Om een conclusie te trekken over het gehele project zullen we, na de conclusies van de verschillende onderdelen, kijken naar de doelstelling waar het project mee is begonnen en in hoeverre het gestelde doel is bereikt. Naast de doelstelling zijn we uiteraard ook aan het werk gegaan met een aantal onderzoeksvragen waar reeds in hoofdstuk 3-7 een conclusie uit is getrokken.

11.1 Doelstelling

De doelstelling zoals die aan het begin van het project is opgesteld, omschrijft (verkort) het volgende:

E.Novation wil kennis opdoen omtrent het gebruik van NoSQL-databases om zo bij toekomstige projecten een geïnformeerde beslissing te maken om deze technologie al dan niet toe te passen. [...] [Vervolgens] zal er een demo-project gemaakt worden om ontwikkelaars met deze toepassing kennis te laten maken met het gebruik van NoSQL in combinatie met CQRS en Event Sourcing.

Terugkijkend op het project is er zeker kennis opgedaan omtrent het gebruik van NoSQL-databases; hier is aan het begin van het project een literatuuronderzoek naar uitgevoerd. De resultaten van dit literatuuronderzoek zijn vastgelegd in het document wat bijgevoegd is als bijlage D. Naast de keuze voor een specifiek NoSQL databasesysteem is er gekeken in welke gevallen er overwogen kan worden om gebruik te maken van een NoSQL databasesysteem en/of Command Query Responsibility Segregation met Event Sourcing.

Ook aan het tweede deel van de doelstelling, het opstellen van een demo-project voor het gebruik van CQRS, lijkt aan het eind van het project voldaan te zijn. Er zijn twee applicaties opgeleverd waarvan er één gebruik maakt van het een NoSQL-opslagsysteem en daarnaast het Axon Framework.

11.2 Onderzoeksvragen

Tot slot resteert de onderzoeksvraag: *‘Hoe kan er binnen E.Novation gewerkt worden met NoSQL databases binnen applicaties ontwikkeld met de CQRS en ES design patterns?’*

Het antwoord op deze vraag is, zoals in hoofdstuk 4-7 geconcludeerd, veelzijdig. Op basis van de deelvragen kan gesteld worden dat E.Novation voor nieuwe applicaties zonder meer gebruik kan maken van de combinatie CQRS, Event Sourcing en MongoDB. Wel zijn er, zoals geconcludeerd in hoofdstuk 5 meerdere gevallen waarin deze combinatie van technologieën mogelijk niet de beste resultaten op zal leveren.

Zoals in hoofdstuk 6 besproken is, zijn er bij gebruik van deze technologieën wel implicaties voor zowel het ontwikkelproces, de ontworpen systemen als de beheerders en ontwikkelaars wanneer gebruik

gemaakt wordt van deze technologie. Daardoor moet er rekening gehouden worden met scholing van medewerkers en is enig onderzoek nodig voordat de toepassing volledig vlekkeloos kan verlopen.

De rol die performance in deze keuze speelt is uiteindelijk door beperkte tijd niet uitgebreid onderzocht. Des al niet te min kunnen we op basis van hoofdstuk 7 stellen dat de performance voldoende lijkt te zijn voor de schaal waarop onze applicaties werken hebben. Zo was er tijdens het testen van de systemen geen merkbaar verschil in performance tussen beide systemen. Dit suggereert dat er voor systemen met beperkte belasting zonder problemen met MongoDB gewerkt kan worden.

Om voor de combinatie Axon en MongoDB te kiezen, moet de functionaliteit van de te ontwikkelen applicatie aansluiten bij de implicaties van het gebruik van beide systemen. MongoDB is op zichzelf een goede allround database en daarmee in de meeste gevallen goed te gebruiken. Axon, daarentegen, brengt voor eenvoudige applicaties veel complexiteit met zich mee. Wanneer een applicatie complexe 'business logic' bevat, kan het gebruik van Axon voordelen bieden.

Samenvattend is het een kwestie van afwegen of Axon en MongoDB bij de te ontwikkelen applicatie passen: *"pick the right tool for the job"* gaat ook voor deze combinatie op.

12 Aanbevelingen

In dit project zijn we tegen een aantal dingen aangelopen die in een vervolgproject anders aangepakt zouden kunnen worden. Enkele van deze aanbevelingen zijn eerder behandeld in het hoofdstuk Aanbevelingen in bijlage E, dit document kan dan ook nageslagen worden voor een meer specifieke beschrijving dan de aanbevelingen in dit hoofdstuk.

12.1 Verbeteringen

Een eerste verbeterpunt is te zorgen voor een grotere hoeveelheid data. Doordat er in ons project geen grote hoeveelheid features zijn geïmplementeerd is er geen significante statistische conclusie uit onze resultaten af te leiden. Wanneer hier bij een vervolgproject wel de focus op komt te liggen dan zal hier een groot aantal datapunten verzameld moeten worden. De focus in dit project lag meer op het belichten van verschillende invalshoeken dan op trekken van conclusies op basis van een grote dataset maar het is uiteraard altijd interessant om zo sterk mogelijke conclusies te kunnen formuleren aan het eind van een onderzoek.

Een tweede punt is te zorgen voor een voldoende afgeschermd en neutrale ontwikkelomgeving. Tijdens ons project zijn we, tijdens ontwikkelen op 'stopwatch', enige malen problemen in de applicaties tegengekomen welke voor vertraging hebben gezorgd waarbij deze vertraging niet altijd goed te scheiden is van de daadwerkelijk aan het systeem toe te wijzen ontwikkeltijd wat de metingen enigszins verstoort.

Een laatste verbeterpunt is het zorgen voor een applicatie welke zich goed leent voor het gekozen pattern of de gekozen technologie zodat de 'proof of concept' - of testapplicatie zo dicht mogelijk bij de uiteindelijke use case van het systeem komt. Door op deze wijze te werken komen mogelijke voor- en nadelen van de twee systemen waarschijnlijk beter tot hun recht.

12.2 Vervolgonderzoek

Door tijdgebrek en/of een overschot aan wensen zijn er tijdens het project ook meerdere punten aan te wijzen die wel interessant zijn om uit te diepen maar waarvoor geen tijd beschikbaar is geweest. De twee punten die hierbij met name naar voren komen zijn het onderzoeken van datamigratie en het uitvoeren van performance testing.

Het onderzoeken van datamigratie heeft hierbij prioriteit, voor het bedrijf is het belangrijk om te weten hoe er omgegaan kan worden met wijzigingen aan het domeinmodel, de business logic en aanverwante zaken. Hierbij kan er voortgebouwd worden op de kennis die al aanwezig is binnen het AZeRa-team van het bedrijf waar er tevens gewerkt wordt met CQRS, hetzij zonder NoSQL database en Event Store.

Tot slot kan ook het uitvoeren van performance testing onder realistische omstandigheden interessant zijn. Hoewel er binnen het bedrijf momenteel nog niet veel tegen de limieten van de beschikbare capaciteit aangelopen wordt is het mogelijk een interessante opdracht om uit te zoeken hoeveel speling hier precies beschikbaar is en of deze mogelijk efficiënter opgevangen kan worden door de applicatie op een NoSQL database te laten draaien.

13 Reflectie

13.1 Algemeen

Als we terugkijken op dit project zijn we een onbekend terrein opgestapt, namelijk het werken binnen een middelgroot bedrijf met professionele collega's met een redelijk 'echte' opdracht. Deze stap in het onbekende heeft goed uitgepakt, we zijn in een prettig bedrijf met leuke collega's en een prettige werksfeer terecht gekomen. We hebben aan het begin wel moeten wennen aan de manier van communiceren met de begeleider en collega's binnen het bedrijf, hier speelt vanzelfsprekend eigen initiatief een grote rol. Na een wat onwennige start zijn we steeds beter bekend geworden met de bedrijfscultuur wat al met al een prettige ervaring is geweest.

Tijdens het project hebben we ook kennis gemaakt met Scrum in de praktijk. Het heeft enige moeite gekost om er de juiste werkwijze op na te houden en een voldoende realistische planning te maken. Na verloop van tijd is dat steeds beter gegaan. Als er niet aan software wordt gewerkt en het plannen gaat via de Scrum-methode zijn we wat gematigder. Hierbij is het lastig in te schatten hoeveel tijd het afronden van bijvoorbeeld een research document kost, dit past niet in alle gevallen prettig in deze werkwijze. Ook dit zal een punt van ervaring zijn, en bij meer gebruik van deze methodiek uiteindelijk soepeler verlopen.

13.2 Maarten van Meijeren

Ik kijk met een positief gevoel terug op de tijd die we bij E.Novation door hebben gebracht. Het was eerst even wennen aan de nieuwe werkplek, de collega's en de 'normale' werkdag van negen tot vijf. Echter ben ik daar snel aan gewend geraakt en het bleek een leuke werkplek te zijn. Een werkplek waar veel kennis aanwezig is op verschillende disciplines en bereidheid om elkaar te helpen, kortom je voelt je welkom en gewaardeerd.

De grootste verandering met andere projecten was toch het werken in Scrum, elke dag een korte meeting werkt verhelderend en stimulerend. Daarnaast is er nauw contact en is de drempel laag om aan te geven wat er dwars ligt of andere opmerkingen te plaatsen.

Ook vind ik het leuk dat we een grote organisatie van binnen hebben kunnen meemaken. Al hadden we zelf een klein scrumteam, om je heen gebeurt genoeg wat in grotere teams plaatsvindt. Samen is het Scrum proces goed verlopen, dat kan ook zijn omdat we bijna altijd naast elkaar hebben gewerkt. Hierdoor is het eenvoudig even te overleggen, wat denk ik ook positief heeft gewerkt.

Niet alles is verlopen zoals we hadden gehoopt, ik heb in ieder geval veel geleerd van het proces dat we tijdens dit project hebben doorlopen. Onderschatting van de tijd die taken kostten is wel eens voorgekomen. Maar niet alleen het procesmatig, ook inhoudelijk heb ik veel opgestoken. Het is ook interessant om te zien welke technieken en methoden er binnen het bedrijf gebruikt worden en waarom. Zo aan het einde gekomen concludeer ik dat ik met plezier aan het project heb gewerkt en ik ben blij met het resultaat

13.3 Yorick Holkamp

Nu het moment is aangekomen om terug te kijken op het project bij E.Novation kijk ik met een zeer positief gevoel terug op de afgelopen maanden. Het bedrijf kent een erg prettige en zeer informele werksfeer; de deur van iedere kamer op de afdeling staat altijd open en de collega's zijn altijd bereid om te hulp te schieten.

Naast de ervaring met E.Novation is het ook zeer leerzaam geweest om binnen een echt IT-bedrijf met een grote hoeveelheid werknemers te werken en te zien hoe er bij de "echte" bedrijven software wordt ontwikkeld. Dit in contrast met de kleinere projecten die ik als freelance-opdrachten uit heb gevoerd naast mijn studie.

Een van de dingen die mij, met name in het begin, wel vrij zwaar is gevallen was het 'nine to five'-werken binnen een kantooromgeving. Zo ben ik zelf niet bepaald een ochtendmens en was ik vooraf vrij sceptisch over de efficiëntie van deze werkwijze. In de praktijk bleek echter 'Het Nieuwe Werken' vrij gangbaar binnen E.Novation en waren er voldoende mogelijkheden om, uiteraard in overleg, flexibel om te springen met de gehanteerde werktijden of zelfs thuis aan de slag te gaan.

Tijdens de afgelopen maanden is dan ook gebleken dat, hoewel je op kantoor niet altijd de meest productieve uren van de dag door kunt maken, het wel stimulerend en motiverend kan werken wanneer je samen met gedreven collega's op één kamer aan het werk bent.

Tot slot werkt ook de Daily Standup Meeting als onderdeel van het Scrum-proces prettig doordat je gedwongen wordt om in de eerste plaats na te denken over wat je de komende dag gaat doen en daarnaast inzicht krijgt in wat je collega's en jijzelf zoal hebben gedaan de voorgaande werkdag.

Al met al kijk ik terug op een geslaagde stage waar ik zeker een hoop nieuwe ervaringen mee heb opgedaan waarmee mijn blik op zaken als het werken vanaf een kantoorlocatie en in grotere ontwikkelteams zeker positief is bijgesteld

14 Literatuurlijst

In deze literatuurlijst zijn niet alle bronnen die in dit project zijn gebruikt weergegeven. De bronnen die meer betrekking hebben op een specifiek ander document zijn vermeld in het betreffende document. Deze documenten zijn bijgevoegd in de bijlagen.

A. Lakshman, 'Cassandra – A structured storage system on a P2P Network', 25 augustus 2008,
http://www.facebook.com/note.php?note_id=24413138919

J. Long, 'A Discussion with Allard Buijze on CQRS with the Axon framework', 24 maart 2010,
http://www.infoq.com/articles/cqrs_with_axon_framework

A. Popescu, 'Cassandra @ Twitter: An Interview with Ryan King', 23 februari 2010,
<http://nosql.mypopescu.com/post/407159447/cassandra-twitter-an-interview-with-ryan-king>

D.J. DeWitt, A. Floratou, J. M. Patel, N. Teletia, D. Zhang, 'Can the elephants handle the NoSQL onslaught?'
<http://pages.cs.wisc.edu/~floratou/SQLvsNoSQL.pdf>

AxonFramework,
<http://axonframework.org/>

Junit,
<http://www.junit.org/>

Mockito,
<http://code.google.com/p/mockito/>

15 Bijlagen

In dit eindverslag zijn de meest relevante opgeleverde documenten die tijdens het project zijn geschreven opgenomen ter informatie en ter toelichting op de inhoud van het eindverslag.

15.1 A. Opdrachtschrijving

In deze bijlage staat de opdrachtschrijving zoals deze initieel is overlegd.

15.2 B. Oriëntatieverslag

In het oriëntatieverslag zijn de probleem- en doelstelling uitgewerkt, zijn de vereisten en wensen van het bedrijf uitgewerkt en is er een probleemanalyse uitgevoerd.

15.3 C. Plan van aanpak

Het plan van aanpak bevat de gedetailleerde opdrachtschrijving, de geplande aanpak, projectinrichting en de planning omtrent kwaliteitsborging.

15.4 D. Research Document NoSQL Databases

Deze bijlage bevat het Research Document wat is opgesteld naar aanleiding van het uitgevoerde literatuuronderzoek naar de verschillende NoSQL databasesystemen waarbij er aan de hand van de gestelde kwaliteitseisen een ‘best fit’ is gekozen.

15.5 E. Implementatieanalyse

In dit document worden de verkregen metingen en ervaringen die tijdens de implementatie van de twee applicaties besproken.

15.6 F. Gebruik van Axon en MongoDB

In dit analysedocument wordt gekeken naar de toepasbaarheid van CQRS, Event Sourcing en MongoDB binnen verschillende applicaties.

Bijlage A: Opdrachtomschrijving

Opdrachtomschrijving

De afdeling Application Development ontwikkelt o.a. web en messaging applicaties, welke in het eigen computercentrum in productie worden genomen. Klanten nemen het product als dienst af, niet als software pakket (SaaS principe).

De meeste van deze toepassingen zijn gebouwd volgens een n-tier architectuur waarin de domeinobjecten via een ORM framework zoals Hibernate worden gepersisteerd in een relationele database.

Recentelijk wordt ook ontwikkeld volgens het Command Query Responsibility Segregation pattern (CQRS) in combinatie met Event Sourcing. De persistentie wordt dat geval verzorgd door een Event Store.

Opdracht: het onderzoeken van de toepassingsmogelijkheden van NoSQL databases binnen de huidige set van SaaS-diensten in het portfolio van E.Novation. In de oriëntatiefase worden de beschikbare NoSQL oplossingen aan de hand van een literatuurstudie vergeleken op basis van relevante functionaliteit, externe kwaliteitsattributen, zoals robuustheid, prestaties, schaalbaarheid en interne kwaliteitsattributen, zoals analyseerbaarheid, aanpasbaarheid, testbaarheid.

Vervolgens wordt een proof-of-concept uitgewerkt op basis van een van de bestaande toepassingen van E.Novation of een representatie fictief voorbeeld. Hieraan zullen diverse tests worden onderworpen om bovengenoemde kwaliteitsattributen in de praktijk te meten. Daarnaast zal een statische kwaliteitsanalyse plaatsvinden (eventueel deels via de Software Improvement Group).

Voorwaarden

De opdracht wordt uitgevoerd op de afdeling Application Development van E.Novation te Capelle. We werken bij E.Novation volgens de Scrum methodiek, ook bij afstudeerprojecten hebben we hier goede ervaring mee. Het Scrum team wordt gevormd door 2 studenten en een begeleider van E.Novation, die een dubbelrol van Scrum Master en Product Owner op zich neemt. De werkzaamheden kunnen parttime worden uitgevoerd, met een minimum 24 uur per week, met dien verstande dat beide studenten op dezelfde dagen werken.

Contact

Contactpersoon bij E.Novation is Bastiaan Bakker, bastiaan.bakker@novation.nl, 010-2881600.

Bijlage B: Oriëntatieverslag

Bachelorproject 2012

Voor de Bachelor Technische Informatica, TU Delft

Maarten van Meijeren, Yorick Holkamp

In samenwerking met *E.Novation*

Begeleiders

Bastiaan Bakker (E.Novation, afdeling Application Development)

Jan Hidders (TU Delft, afdeling Web Information Systems)



Inhoudsopgave

1.	Inleiding	3
2.	Probleemstelling	4
3.	Doelstelling	4
4.	Opdrachtformulering	5
5.	Kwaliteitseisen	5
5.1.	Primaire eisen	5
5.2.	Secundaire eisen	6
6.	CQRS	6
6.1.	N-tier	6
6.2.	CQS	7
6.3.	CQRS	7
7.	Event Sourcing	8
8.	CQRS/ES	8
9.	Scrum	9
9.1.	Sprints	9
9.2.	Taakverdeling	9
9.3.	Werkverdeling	9
9.4.	Conclusie	10
10.	NoSQL databasesystemen	10
10.1.	Definitie NoSQL	10
10.2.	Fundamentele principes	10
10.2.1.	ACID	11
10.2.2.	CAP	12
10.2.3.	BASE	12
10.2.4.	Afweging 'C', 'A' en 'P'	13
10.3.	Verschillende typen NoSQL databases	14
10.3.1.	Key-value stores	14
10.3.2.	Document stores	14
10.3.3.	Wide column / distributed peer stores	14
10.3.4.	XML	14
10.3.5.	Object	14
11.	Bibliografie	15

1. Inleiding

E.Novation¹ is een bedrijf dat verschillende IT-diensten verleent, zowel maatwerk als standaardoplossingen en business consulting. Het bedrijf is hiervoor onderverdeeld in specifieke business units, zo verzorgt Managed Services bijvoorbeeld een set aan diensten volgens het Software-as-a-Service-principe.

Binnen deze set diensten speelt de E.Berichtendienst een grote rol, hiermee worden verschillende beveiligde communicatiesystemen aangeboden zoals ENS, welke onder andere gebruikt wordt door TNT Post, scheepvaart-communicatiedienstverlener Portbase en China Shipping.

In het algemeen werkt E.Novation hierbij voornamelijk voor de overheid en semioverheid. Inmiddels is er een nieuw project 'from scratch' bezig om zorgverleners en verzekeraars onderling te laten communiceren over dienstverlening binnen het kader van de AWBZ. Hierbij is gekozen om op basis van de Command Query Responsibility Segregation (CQRS) en Event Sourcing patterns te gaan ontwikkelen.

Wat hierbij binnen het bedrijf naar boven is gekomen is dat het wellicht voordelen zou kunnen bieden op meerdere vlakken wanneer dit gecombineerd wordt met een NoSQL-database als Event Store of query database.

¹ (E.Novation, enovation.nl)

2. Probleemstelling

Onze probleemstelling is: “Hoe kan er binnen E.Novation gewerkt worden met NoSQL databases binnen applicaties ontwikkeld met de CQRS en ES design patterns?”

Om deze probleemstelling te kunnen beantwoorden zullen wij proberen de volgende deelvragen te beantwoorden:

1. Welke NoSQL-database sluit het beste aan op de interne en externe kwaliteitseisen? (Meer over kwaliteitseisen, zie hoofdstuk 5)
2. Welke mogelijkheden en limitaties geeft een NoSQL database ten opzichte van de huidige SQL-oplossing?
 - a. Welke voordelen geeft de toepassing van NoSQL als Event Store binnen ES?
 - b. Welke voordelen geeft de toepassing van NoSQL als view database?
 - c. Welke toepassingen profiteren weinig of niet van een NoSQL-oplossing?
3. Wat zijn de gevolgen voor het ontwikkelproces van E.Novation?
 - a. Moet het ontwikkelproces gewijzigd worden?
 - b. Hoe kunnen systeemwijzigingen op later moment doorgevoerd worden?
 - c. In hoeverre moeten de applicatiebeheerders opgeleid worden?
 - d. Verhoogt het gebruik van CQRS/ES de efficiëntie van de ontwikkeling?
4. Wat zijn de performance-grenzen wanneer er gewerkt wordt met een NoSQL database?
 - a. Hoe ver reikt de performance van een enkele server?
 - b. Hoeveel winst levert het toevoegen van extra servers aan de database cluster?

3. Doelstelling

E.Novation wil kennis opdoen omtrent het gebruik van NoSQL-databases om zo bij toekomstige projecten een geïnformeerde beslissing te maken om deze technologie al dan niet toe te passen. Momenteel is er nog geen keuze gemaakt voor een specifiek NoSQL databasesysteem. Om tot een keuze te komen zullen wij een literatuuronderzoek uitvoeren en de beschikbare systemen toetsen aan de hand van de benodigde functionaliteit en gehanteerde kwaliteitseisen van het bedrijf.

Nadat bekend is welke DBMS het best aansluit op de kwaliteitseisen van E.Novation zal er een demoproject gemaakt worden om ontwikkelaars kennis met deze toepassing te laten maken met de toepassing van NoSQL in combinatie met CQRS en Event Sourcing. Hierbij kan er tevens een ‘sandbox’ aangeboden worden voor eventuele toekomstige proof of concepts.

4. Opdrachtformulering

Het onderzoeken van de toepassingsmogelijkheden van NoSQL databases binnen de huidige set van SaaS-diensten in het portfolio van E.Novation. In de oriëntatiefase worden de beschikbare NoSQL oplossingen aan de hand van een literatuurstudie vergeleken op basis van relevante functionaliteit, externe kwaliteitsattributen zoals robuustheid, prestaties, schaalbaarheid en interne kwaliteitsattributen zoals analyseerbaarheid, aanpasbaarheid, testbaarheid.

Vervolgens wordt een proof-of-concept uitgewerkt op basis van een van de bestaande toepassingen van E.Novation of een representatief fictief voorbeeld, waarbij gebruik gemaakt wordt van het Command Query Segregation (CQRS) pattern in combinatie met Event Sourcing. Hieraan zullen diverse tests worden onderworpen om bovengenoemde kwaliteitsattributen in de praktijk te meten. Daarnaast zal een statische kwaliteitsanalyse plaatsvinden (eventueel deels via de Software Improvement Group).

5. Kwaliteitseisen²

De te hanteren kwaliteitseisen voor de potentiële NoSQL databasesystemen zijn tweeledig, zo is er een set punten die primair vereist zijn en enkele punten en enkele zaken die bekeken dienen te worden maar niet van doorslaggevend belang zijn. Deze punten zijn gegroepeerd per onderwerp.

5.1. Primaire eisen

Om in aanmerking te komen voor gebruik binnen het project zal een databasesysteem in ieder geval op de volgende punten voldoende hoog moeten scoren.

- Functionaliteit
 - Te gebruiken met een applicatie geschreven in Java.
 - Beschikt over voldoende functionaliteit om in te passen in toepassingen binnen E.Novation.
- Betrouwbaarheid
 - Foutbestendig bij uitval van servers.
 - Herstelbaar na fouten en uitval van servers.
- Onderhoudbaarheid van de databaseserver
 - Mogelijkheid tot het analyseren van het gedrag van de server, een openbare broncode is hierbij een pré om gedrag te kunnen verklaren.
 - Mogelijkheid tot het aanpassen van de software en deze wijzigingen kunnen testen op neveneffecten in de overige code.
 - Beschikbaarheid van tooling om de status van servers bij te houden.
 - Voldoende stabiel voor een productieomgeving, hierbij gelet op zowel eventuele grote wijzigingen in de API, kritieke bugs en voldoende toekomstperspectief.

² (Testconsultancy-Groep, 1998)

5.2. Secundaire eisen

Naast de primaire eisen zijn er tevens een aantal zaken die zeer wenselijk zijn maar niet de boventoon voeren, ook deze punten zijn gegroepeerd per relevante factor.

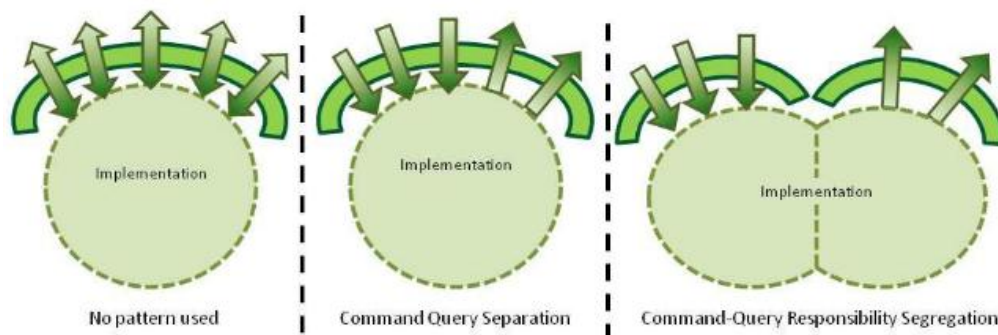
- **Bruikbaarheid**
 - Voldoende begrijpelijke concepten voor ontwikkelaars.
 - De leerbaarheid van de systemen vormt geen grote drempel.
 - Beheerbaarheid van productieservers.
- **Vervangbaarheid**
 - De mogelijkheid om op enig moment zonder vergaande applicatiewijzigingen te kiezen voor een alternatief databasesysteem.
- **Efficiëntie**
 - Geen buitensporig middelenbeslag ('resource utilisation').
 - Schaalbaar om in mogelijke toekomstige groei te kunnen voorzien.

6. CQRS

Uit de opdrachtomschrijving en bijbehorende onderzoeksvragen en doelstelling van het project blijkt dat er in de op te leveren producten gewerkt zal gaan worden door middel van het CQRS-pattern. In dit hoofdstuk zal kort besproken worden wat dit inhoud en welke voordelen dit met zich mee kan brengen. Om dit te doen zal eerst kort een blik geworpen worden op de traditionele manier van bouwen om vervolgens over te stappen naar de verschillen bij het gebruik van CQRS.

6.1. N-tier³

Veel grote client-server systemen worden traditioneel multi-tier (ook wel N-tier) opgebouwd, dit betekent dat voor elke verantwoordelijkheid van de applicatie er een aparte laag (tier) wordt toegevoegd. Een probleem wat hierbij voor kan komen is dat in een applicatie veel complexiteit ontstaat door deze gelaagde structuur doordat voor alle functionaliteiten en verantwoordelijkheden eigen tiers toegevoegd worden welke onderling informatie door moeten spelen.



Figuur 1 (van der Arend, 2010)

³ (van der Arend, 2010)

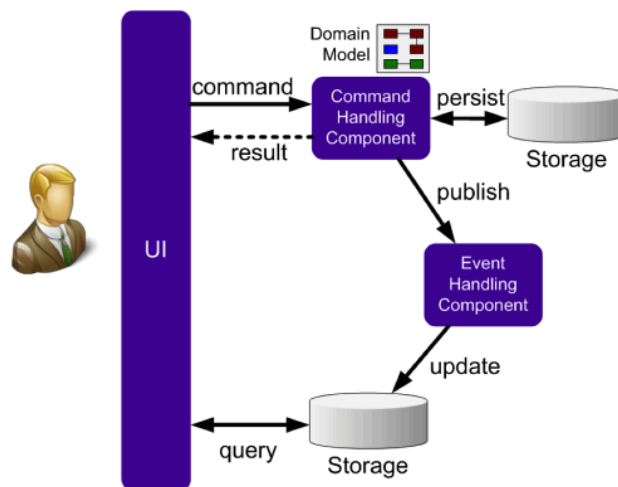
6.2. CQS⁴

Een mogelijk opstapje naar het oplossen van deze problematiek is het Command Query Separation (CQS) idee van Meyer (Meyer, 1988). CQS stelt (zie figuur 1) dat er voor lezen en schrijven verschillende methoden gebruikt dienen te worden om het systeem op die manier zover mogelijk te ontdoen van 'bijwerkingen' bij het opvragen van bepaalde gegevens. Zoals dit ook wel verwoord wordt, *"asking a question should not change the answer"*.

6.3. CQRS⁵

Command and Query Responsibility Segregation (CQRS) gaat hierbij iets verder door ook voor de verschillende acties verschillende interfaces aan te bieden en op die manier het geheel nog verder van elkaar los te halen. Hiermee kan er op transparante manier gezorgd worden voor losse afhandeling van de verschillende acties en er op die manier gezorgd worden voor een verdere 'separation of concerns'. Naast de voordelen op het gebied van SoC kan er, zoals in veel van de implementaties van CQRS gebeurt, tevens gewerkt worden met losse databases voor queries en commands.

Een relatief grote stap hierbij is het accepteren dat data die wordt getoond aan een gebruiker per definitie niet up-to-date is, de data kan immers tegelijk in parallel aangepast worden. Wanneer dit als 'fact of life' gezien wordt kan er ook voor worden gekozen om de query-database (de onderste 'storage' in figuur 2) periodiek of asynchroon bij te werken aan de hand van de data in de command-database (bovenste 'storage' in figuur 2). In dat geval kan men spreken over een systeem met 'eventual consistency'; de data zal uiteindelijk voor alle gebruikers consistent zijn maar kan op tussengelegen momenten verschillen.



Figuur 2

Wanneer we vervolgens kijken naar de ondergelegen indeling zoals te zien in figuur 2 dan wordt de algehele structuur inzichtelijker. De gebruiker kan via de UI commando's doorgeven aan de 'command handling component' die in de permanente storage wordt opgeslagen. Daarnaast worden de wijzigingen in de vorm van events doorgegeven aan de 'event handling component' welke deze asynchroon zal doorvoeren in de query-database.

Zodra de gebruiker vervolgens informatie op wil vragen, wordt er een query verstuurd naar de query-database, waarna de gebruiker een versie van de data te zien zal krijgen.

⁴ (Fowler, CommandQuerySeparation) (Meyer, 1988)

⁵ (Abdullin, 2010) (Dahan, 2009) (Šaltys, 2010) (Young, CQRS, Task Based UIs, Event Sourcing agh!, 2010) (Fowler, CQRS, 2011)

Een voordeel van deze aanpak is dat er met de losse query-database relatief eenvoudig voor hogere query-capaciteit gezorgd kan worden. Dat kan door hier te werken met database die de data bijvoorbeeld in geheugen bewaart of horizontaal goed schaaft. Hiermee wordt het eenvoudig mogelijk om voor de query- en command-database zelfs andere databasesystemen te gebruiken. Een mogelijke invulling is bijvoorbeeld voor de query-database een systeem te kiezen met hoge leescapaciteit en voor de command-database systeem met hoge schrijfcapaciteit, een andere is om voor de opslag te kiezen voor een database die efficiënt de objecten op kan slaan en aan de kant van de query-database voor een systeem als PostgreSQL om zoekacties in bepaalde tekstvelden efficiënt aan te bieden.

Kortom, CQRS biedt enkele voordelen om de algemene structuur van de software eenvoudiger en verder opgesplitst op te stellen. Daarnaast hoeft er niet per se gedacht te worden aan één relationele database om alles af te handelen maar mogelijk zelfs aan verschillende read- en write-databases. Beide geoptimaliseerd voor toepassing waarvoor ze gebruikt worden. Wanneer CQRS dan gecombineerd wordt met events binnen het systeem dan is het te combineren met Event Sourcing, waarover meer in het volgende hoofdstuk.

7. Event Sourcing⁶

Naast CQRS wordt er binnen een project van E.Novation tevens gewerkt met het Event Sourcing pattern. In gangbare systemen wordt vaak per domain object de huidige state opgeslagen in een database, waarna deze objecten ingeladen kunnen worden door middel van een Object Relational Mapper (ORM) als Hibernate die de opgeslagen data omzet naar objecten binnen het programma.

Een nadeel van deze aanpak is dat informatie over hoe het object in de huidige toestand is gekomen op deze manier verloren gaat. Dit kan nadelig zijn omdat deze informatie van pas kan komen bij het oplossen van fouten tijdens de uitvoer van het programma, ondersteuning aan klanten of gewoonweg om bij te houden welke functies binnen het programma veel gebruikt worden.

Event Sourcing is een pattern waarmee deze informatie bewaard kan blijven, door de objecten in de database op te slaan als een sequentie van events. Wanneer vervolgens een object opgevraagd wordt kunnen deze events na elkaar toegepast worden op een 'kaal' object om zo weer tot de huidige state van het object te komen.

8. CQRS/ES⁷

Wanneer er gekozen wordt voor CQRS in combinatie met een Event Driven Architecture (EDA), zoals het geval is binnen E.Novation, ontstaat er een combinatie die interessant kan zijn om de geschiedenis van Event Sourcing en de onderverdeling van CQRS beide te benutten.

Binnen CQRS wordt er, zoals eerder uitgelegd in hoofdstuk 5, gewerkt met een los concept van een view- en query-database. Om hierbij Event Sourcing toe te passen zal de command-database vervangen

⁶ (Young, Why use Event Sourcing?, 2010) (Young, CQRS, Task Based UIs, Event Sourcing agh!, 2010) (Fowler, Event Sourcing, 2005)

⁷ (van der Arend, 2010) (Long, 2010) (Young, CQRS and Event Sourcing, 2010)

moeten worden door een 'event store', een opslagsysteem voor de geordende set events die tot het object leiden. Deze verandering zal voor het systeem in verdere mate weinig implicaties hebben wanneer het systeem reeds event driven opgesteld is.

9. Scrum⁸

Binnen E.Novation wordt er volgens het Scrum-principe ontwikkeld. Scrum is een vorm van Agile Programming, wat een alternatief is voor de traditionele ontwikkelingsmethoden als het 'watervalmodel'. Scrum is een principe wat gebaseerd is op een viertal beweringen die samen het handvest worden genoemd. Een voorbeeld van één van deze bewering: "Het is beter om goed om te gaan met verandering dan vast te zitten aan een vooropgezet plan."

9.1. Sprints

Kenmerkend voor Scrum zijn de zogenaamde 'sprints' van ontwikkeling. Deze sprints zijn iteraties van ontwikkeling, als het ware deelprojecten in een project. Een sprint duurt normaliter rond de twee tot vier weken, bij E.Novation houden ze een sprint van ongeveer 3 weken aan. Het is de bedoeling dat in elke sprint een werkend product wordt opgeleverd. De Sprint is onderverdeeld in nog kleinere iteraties van een dag. Elke dag wordt er vergaderd door het team waarbij er een planning voor die dag opgemaakt wordt.

9.2. Taakverdeling

Hoe ziet een Scrum team er dan uit? Degene die nauw contact heeft met de klant is in de Scrumwereld de Product Owner, een representatie van de klant in het bedrijf. Aan het hoofd van een team staat de ScrumMaster, die belegt elke dag een 'Daily Standup Meeting'. Daarin komt het Development Team samen. Dit team maakt het product. In ons geval is de begeleider, Bastiaan Bakker, de ScrumMaster en tegelijk de Product Owner. Nu is dit niet de meest wenselijke situatie, aangezien de Product Owner zich dan teveel inlaat met de ontwikkeling en de manier waarop er ontwikkeld wordt. De Product Owner zou zich namelijk alleen zorgen moeten maken over het feit dat het product werkt naar zijn eisen of niet.

9.3. Werkverdeling

De punten die in een sprint moeten worden gerealiseerd komen uit de 'backlog' dat is een lijst van requirements die op prioriteit zijn gesorteerd. In een sprint wordt een deel van het product opgeleverd, dat moet wel een werkend product zijn. De totale backlog wordt onderverdeeld in een kleinere backlog per sprint. Zo wordt dus bepaald wat er gedaan kan/moet worden in een sprint-iteratie. Per dag wordt er een standup-meeting gehouden, hier komen de teamleden samen om te bespreken wat ze afgelopen dag gedaan hebben, komende dag gaan doen en de problemen die zij tegen zijn gekomen. Op deze manier wordt iedereen betrokken gehouden en doet het team de opdracht samen. De teamgeest wordt op deze manier getracht hoog te houden. Daarnaast wordt het project strak in de gaten gehouden, zo kun je snel anticiperen op onvoorziene situaties.

⁸ (Beck, 2001) (Cohn) (Kniberg, 2007)

9.4. Conclusie

De Scrum-methode is een andere methode dan de watervalmethode. Het kan zo zijn dat het project daarom iets anders verloopt dan de meeste reguliere bachelor projecten. De ervaringen van het bedrijf met Scrum en afstudeerders zijn erg goed.

10. NoSQL databasesystemen⁹

Om een algemeen beeld te krijgen van de verschillende NoSQL databasesystemen zijn we kort ingegaan op de verschillende beschikbare systemen. Hierbij zullen we eerst kort stilstaan bij wat NoSQL nu daadwerkelijk betekent om vervolgens kort te bespreken welke fundamentele principes ten grondslag liggen aan deze en andere databasesystemen. Tot slot zullen we kort aanstippen welke verschillen typen systemen er te onderscheiden zijn, waarmee onze zoektocht naar de ideale oplossing beperkt zou kunnen worden tot enkele typen.

10.1. Definitie NoSQL

Er wordt in de literatuur geen eenduidige definitie van NoSQL gehanteerd, er wordt vaak meer gewerkt met een definitie in de trend van de 'grootste gemene deler'. Zo zijn NoSQL-databases in het algemeen non-relationeel en ligt de focus op specifieke usecases. Er zijn dus verschillende interpretaties maar een definitie is als volgt:

Volgens (Barbosa, 2011) zijn NoSQL-databases "hoofdzakelijk non-relational, distributed, open-source en horizontal scalable". Als een database non-relational is, wil dat zeggen dat de database niet gebaseerd is op een (traditioneel) relationeel databasemodel. Dat het distributed is betekent dat de data verspreid, over meerdere servers opgeslagen en bekeken kan worden. Het begrip open source moge duidelijk zijn, tot slot houdt "horizontal scalability" in dat de performance opgehoogd kan worden door meer servers toe te voegen.

10.2. Fundamentele principes

De principes BASE en CAP zijn aan databases verbonden waarbij BASE is een soepelere vorm van het ACID principe is. Het ACID principe zorgt ervoor dat in relationele databases de transacties betrouwbaar worden uitgevoerd. BASE is een 'lossere' tegenhanger van ACID voor grote systemen die de ACID-garanties, die hierna besproken zullen worden, niet kunnen garanderen. CAP hiernaast is een theorie over gedistribueerde systemen in het algemeen, daar heeft NoSQL ook mee te maken.

⁹ (Rees) (Barbosa, 2011) (Harrison)

10.2.1. ACID

Het ACID principe bestaat uit vier verschillende punten (Barbosa, 2011). Deze vier eigenschappen hebben als doel de betrouwbaarheid van de database transactie te garanderen in elke willekeurige situatie. ACID staat voor de volgende zaken:

- Atomic: ondersteuning voor transacties waarbij als één van de operaties niet werkt dan worden alle wijzigingen, die gedaan zijn binnen de transactie, teruggedraaid.
- Consistent: wanneer er een fout optreedt bij het uitvoeren van een transactie moet de database altijd in een geldige staat terecht komen.
- Isolation: transacties die tegelijk worden uitgevoerd mogen geen effect hebben op de resultaten van elkaar.
- Durability: zodra een transactie is uitgevoerd moet deze staat van het systeem bewaard blijven, ongeacht fouten, crashes of uitval van de server.

Wanneer een systeem aan al deze eisen voldoet dan kan er met zekerheid gesteld worden dat de database onder alle omstandigheden een verwacht resultaat zal geven. De nadruk van ACID ligt op de betrouwbaarheid van de transacties. Een voorbeeld waar betrouwbaarheid zeer hoge prioriteit heeft is in een financiële instelling. Er mag tijdens transacties geen geld bijkomen of verdwijnen omdat er iets in de database mis gaat. Aan de hand van een geld transactie kunnen we de vier punten toelichten (IBM, 2012).

10.2.1.1. Atomic

Een atomic transactie zorgt ervoor dat binnen één transactie alle wijzigingen gezien worden als één. Als er één wijziging mislukt, mislukt de hele transactie. Concreet, als tijdens het overmaken van geld, geld van rekening1 wordt afgehaald moet het ook op rekening2 worden bijgeschreven. Atomic zorgt ervoor dat er niet alleen kan worden afgeschreven of bijgeschreven.

10.2.1.2. Consistency

Consistency heeft als doel dat de data altijd consistent is, voor de transactie en na de transactie. In het geval van een overboeking zorgt consistency ervoor dat de hoeveelheid geld voor en na de transactie gelijk is. Er kan niet meer afgeboekt worden dan dat er opgeboekt wordt.

10.2.1.3. Isolation

Isolation heeft betrekking op het wijzigen van data door meer dan één transactie. Indien er meer dan één transactie de data wil veranderen kan dat gevolgen hebben voor de consistentie van de data. Omdat de data hetzelfde moet blijven zorgt Isolation ervoor dat de data maar door één transactie tegelijk gewijzigd kan worden. Als we dat betrekken op een banktransactie zal een transactie in gewijzigde staat op ongewijzigde staat beschikbaar zijn. Het is in ieder geval niet zichtbaar tijdens de transactie.

10.2.1.4. Durability

Nadat de transactie is uitgevoerd heeft durability de taak om te zorgen dat de transactie niet wordt teruggedraaid. Ook niet als het systeem crasht of de stroom uitvalt. Als er eenmaal geld is overgemaakt

mag dit niet zomaar ongedaan worden gemaakt. Durability heeft de taak om ervoor te zorgen dat dit niet gebeurt.

Het nadeel van deze set eisen is dat ze niet te combineren zijn met gedistribueerde systemen die hoge performance moeten leveren¹⁰ door de hoeveelheid extra regelwerk om deze garanties af te kunnen dwingen. Daarnaast zal de applicatie moeten samenwerken met verschillende bestaande systemen waarvan deze punten niet gegarandeerd kunnen worden.

10.2.2. CAP¹¹

Dit is tevens wat het CAP theorem (Harrison) stelt, namelijk dat het binnen gedistribueerde systemen niet mogelijk is om meer dan twee van de volgende punten te verzorgen:

- Consistency, alle nodes zien dezelfde data op hetzelfde moment.
- Availability, je kunt altijd je data wegschrijven en opvragen naar het systeem, of het nu succesvol is of niet. Met andere woorden: het systeem is altijd beschikbaar.
- Partition tolerance, het systeem blijft functioneren bij falen van de communicatie tussen verschillende servers voor arbitraire tijd, of verlies van gegevens.

In veel gevallen van een gedistribueerd systeem is de Partition tolerance een vereiste. Deze tolerance zorgt ervoor dat het systeem bij gedeeltelijk falen blijft werken. Als dit niet zo is, dan is het systeem zo sterk als de zwakste schakel, vooral met grote systemen is dit niet wenselijk.

Bij ACID is consistency belangrijk, dat betekent dat de Availability niet verzorgd zou kunnen worden. Dat is voor een groot systeem dat veel gebruikers heeft niet handig, dan zullen veel gebruikers last hebben van vertragingen die ontstaan door het onmiddellijk consistent maken van de database. Voor de gedistribueerde systemen bestaat het BASE principe.

10.2.3. BASE¹²

Voor gedistribueerde databasesystemen is dan ook een tegenhanger van ACID geformuleerd, namelijk BASE:

- Basically Available, het systeem lijkt altijd te functioneren en heeft zoveel mogelijk tijd zoveel mogelijk data beschikbaar.
- Soft state, het systeem springt flexibel om met de consistentie van data in de database verspreid over verschillende nodes.
- Eventually consistent, uiteindelijk komt het systeem in een consistente state terecht maar dit zal soms pas zijn nadat alle wijzigingen tussen verschillende nodes gesynchroniseerd zijn.

Deze ontwerpkeuzes zijn logischerwijs niet zonder consequenties. De aanpak van 'basically available' betekent dat het systeem zoveel mogelijk beschikbaar is. Omdat met CAP de partition tolerance en Availability bezet zijn heeft dit gevolgen voor de consistency. Een consequentie is dat in veel gevallen systeemfouten niet op lokaal niveau opgemerkt hoeven te worden in de vorm van failures.

¹⁰ (Wikipedia, 2012) (Gilbert & Lynch) (Brewer, 2000)

¹¹ (Brown, 2009)

¹² (Pritchett, 2008) (Roe, 2012)

‘Soft state’ en ‘Eventual consistency’ geven de beperking dat er in het ontwerp van applicaties en processen rekening mee moet worden gehouden dat een instantie van de applicatie niet altijd het gehele overzicht paraat zal hebben. De data is niet met een ‘harde’ garantie up-to-date.

10.2.4. Afweging ‘C’, ‘A’ en ‘P’

De vraag is welke van CAP garanties het beste losgelaten kan worden. Aan elk van de punten die niet worden gebruikt in het systeem zitten consequenties.

Als we een systeem hebben wat uit meer delen bestaat en er wordt geen partition tolerance gebruikt, functioneert het geheel niet meer als er een gedeelte mee stopt. Een manier om dit uit te sluiten is om alles op één machine te laten functioneren. Hier zit het nadeel aan dat de schaling erg beperkt is. CAP is van toepassing bij gedistribueerde systemen, dat zijn bijna altijd zeer grote systemen die uit meerdere delen bestaan. De keten is dan zo sterk als de zwakste schakel. Partition tolerance is dan ook bijna altijd van toepassing binnen een gedistribueerd systeem.

Beperkte Availability heeft als gevolg dat het systeem altijd consistent blijft en partition tolerance intact is. Het voordeel is dat er geen ‘stale reads’ plaatsvinden, dat is het lezen van data die niet up-to-date is en een andere waarde heeft op de master server. Consistentie is dan belangrijk, de beperkte availability geeft wel een hogere latency bij het lezen. Daarnaast een lagere throughput van het lezen. Immers als alle data up-to-date moet zijn, moet dat wel gecheckt worden. Daardoor zit er veel overhead in het controleren en eventueel aanpassen van de data aanwezig in de nodes.

Voor beperkte Consistency gelden eigenlijk de omgekeerde regels. Er zijn wel stale reads mogelijk. De latency en throughput zijn beter. Veel eigenschappen van beperkte consistency geven de beperking dat er in het ontwerp van applicaties en processen rekening mee moet worden gehouden dat een instantie van de applicatie niet altijd het gehele overzicht van recente data paraat zal hebben.

Het is ook afhankelijk wat de toepassing is waar CAP op van toepassing is. Stel dat het een applicatie is die altijd consistent moet zijn, dan is het beter dat het iets langer duurt voordat het compleet is uitgevoerd dan dat het verkeerde resultaten oplevert. Daar zijn de ‘C’ en ‘P’ van toepassing. Zoals eerder gesteld is de ‘P’ bijna altijd van toepassing. Als de ‘A’ dan nodig is, is de ‘C’ bijna vanzelfsprekend ondergeschikt. Een voorbeeld is bijvoorbeeld Twitter, waarbij het niet erg is dat er een bericht iets later de wereld in komt, als het maar gebeurt.

Ook is er een hybride vorm mogelijk, dat verschillende taken waarbij consistency belangrijk zijn anders worden uitgevoerd dan taken waarbij availability belangrijk is. Over het algemeen worden nu vaak de ‘A’ en de ‘P’ toegepast in gedistribueerde systemen. Mocht de ‘C’ erg belangrijk zijn dan is het mogelijk om de balans om te gooien. Het hangt af van de eisen van de toepassing welke balans er aangehouden moet worden. Als het systeem niet altijd consistent is zal er met de ontwikkeling van applicatie ook rekening mee gehouden moeten worden dat niet altijd de meest up-to-date informatie in de een node van de database staat.

10.3. Verschillende typen NoSQL databases

Barbosa (Barbosa, 2011) geeft aan dat NoSQL puur een term is voor een groep databases die geschikt zijn om bepaalde situaties aan te kunnen. Bijvoorbeeld als je snel ongestructureerde data, die een kleine totale omvang heeft, wilt opslaan of juist terabytes aan logs van een groot advertentienetwerk. Elk van deze toepassingen heeft zijn eigen trade-offs om te voldoen aan de specifieke vereisten.

10.3.1. Key-value stores

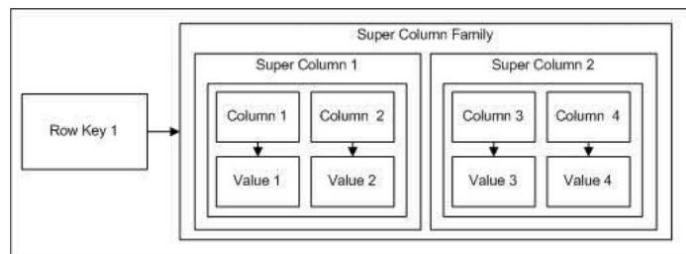
De meest simpele vorm van NoSQL databases zijn mogelijk de key-value stores, hierbij bestaat de database in feite uit een map van keys met bijbehorende values, waarbij de values bytes, strings of in sommige gevallen, zoals bij Redis, ook enkele complexere datastructuren als sets en maps.

10.3.2. Document stores

Een subset van de key-values stores zijn document stores, deze databases werken ook op basis van een key maar beslaan in plaats van eenvoudige typen uit 'documenten' welke gerepresenteerd kunnen worden als een map met verschillende typen data als values.

10.3.3. Wide column / distributed peer stores

De Wide Column Stores werken op basis van het BigTable-principe, hierbij bevat elke row een vaste hoeveelheid kolomverzamelingen, waarbij elke kolomverzameling een flexibel aantal kolommen kan bevatten. Cassandra heeft de mogelijkheid tot Super kolomverzameling in plaats van een normale kolomverzameling. Een Super kolom kan kolomverzamelingen bevatten die ieder op hun beurt er flexibel aantal kolommen kunnen bevatten. in figuur 3 is er een voorbeeld van Cassandra afgebeeld. De rijen allemaal bij elkaar worden de



Figuur 3

database, de keyspace. Een willekeurige andere BigTable zal geen super kolomverzameling ondersteunen.

10.3.4. XML

XML-databases zijn storages die input en output in XML-formaat accepteren en/of deze data ook daadwerkelijk in XML opslaan. Deze data kan dan, door middel van bijvoorbeeld XQuery, doorzocht worden.

10.3.5. Object

Objectdatabase is een vorm van database waarin de data opgeslagen wordt als een object, dat is te vergelijken met de objecten in Object Oriented Programming. Door dit samen te voegen met de applicatie kun je een zogenaamd object-oriented database management system krijgen. Dat kan voor hoge mate van consistentie zorgen, zo zijn de database en applicatie op dezelfde Objecten gebaseerd.

11. Bibliografie

- Abdullin, R. (2010, november 3). *CQRS Architecture and Definitions*. Retrieved maart 2012, from abdullin.com: <http://abdullin.com/journal/2010/11/3/cqrs-architecture-and-definitions.html>
- Barbosa, P. L. (2011, juni 27). Cloud-based data storage. Delft: TU Delft.
- Beck, F. &. (2001). *Manifesto for Agile Software Development*. Retrieved maart 2012, from agilemanifesto.org: <http://agilemanifesto.org/>
- Brewer, A. (2000, juli 19). *Towards Robust Distributed Systems*. Retrieved juli 2012, from www.cs.berkeley.edu: <http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf>
- Brown, J. (2009, januari 11). *Brewer's CAP Theorem*. Retrieved mei 2012, from julianbrowne.com: <http://www.julianbrowne.com/article/viewer/brewers-cap-theorem>
- Cohn, M. (n.d.). *Introduction to Scrum - An Agile Process*. Retrieved maart 2012, from mountaingoatsoftware.com: <http://www.mountaingoatsoftware.com/topics/scrum>
- Dahan, U. (2009, december 9). *Clarified CQRS*. Retrieved maart 2012, from udidahan.com: <http://www.udidahan.com/2009/12/09/clarified-cqrs/>
- E.Novation. (n.d.). Retrieved maart 2012, from enovation.nl: <http://enovation.nl>
- E.Novation. (n.d.). *E.Novation Wiki*. Retrieved maart 2012, from <https://wiki.enovation.nl/index.php/Hoofdpagina>
- Fowler, M. (n.d.). *CommandQuerySeparation*. Retrieved maart 2012, from martinowler.com: <http://martinfowler.com/bliki/CommandQuerySeparation.html>
- Fowler, M. (2011, juli 14). *CQRS*. Retrieved maart 2012, from martinowler.com: <http://martinfowler.com/bliki/CQRS.html>
- Fowler, M. (2005, december 5). *Event Sourcing*. Retrieved maart 2012, from martinowler.com: <http://martinfowler.com/eaDev/EventSourcing.html>
- Gilbert, S., & Lynch, N. (n.d.). Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services. *Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services* .
- Harrison, G. (n.d.). *Survey of Distributed Databases*. Retrieved maart 2012, from dbpedias.com: http://dbpedias.com/wiki/NoSQL:Survey_of_Distributed_Databases
- IBM. (2012, mei 25). *ACID properties of transaction*. Retrieved mei 2012, from IBM.com: <http://publib.boulder.ibm.com/infocenter/cicsts/v3r2/index.jsp?topic=%2Fcom.ibm.cics.ts.productoverview.doc%2Fconcepts%2Facid.html>
- Kniberg, H. (2007). *Scrum and XP from the Trenches: Enterprise Software Development*. InfoQ.

Long, J. (2010, maart 24). *A Discussion with Allard Buijze on CQRS with the Axon framework*. Retrieved maart 2012, from infoq.com: http://www.infoq.com/articles/cqrs_with_axon_framework

Meyer, B. (1988). *Object-Oriented Software Construction*. Upper Saddle River, NJ, USA: Prentice-Hall.

Pritchett, D. (2008, mei 1). *BASE: An Acid Alternative*. Retrieved mei 2012, from queue.acm.org: <http://queue.acm.org/detail.cfm?id=1394128>

Rees, R. (n.d.). *NoSQL, no problem: An introduction to NoSQL databases*. Retrieved maart 2012, from Thoughtworks.com: <http://www.thoughtworks.com/articles/nosql-comparison>

Roe, C. (2012, maart 1). *ACID vs BASE: The Shifting pH of Database Transaction Processing*. Retrieved mei 2012, from Dataversity: <http://www.dataversity.net/acid-vs-base-the-shifting-ph-of-database-transaction-processing/>

Šaltys, Ž. (2010, juli 5). *What is CQRS?* Retrieved maart 2012, from thedeveloperday.com: <http://www.thedeveloperday.com/what-is-cqrs/>

Testconsultancy-Groep. (1998). *Kwaliteitsattribuuten volgens ISO 9126*. Retrieved maart 2012, from testconsultancy-groep.nl: http://www.testconsultancy-groep.nl/index.php/downloads?section=files&task=download&cid=4_5b5d8362a4cc1f3853ffa358b9987470

van der Arend, R. (2010, april 28). *Sogyo Café, schaalbaarheid, CQRS*. Retrieved maart 2012, from slideshare.net: <http://www.slideshare.net/rickvanderarend/een-introductie-in-cqrs>

Wikipedia. (2012). *CAP Theorem*. Retrieved juli 2012, from Wikipedia: http://en.wikipedia.org/wiki/CAP_theorem

Young, G. (2010, februari 13). *CQRS and Event Sourcing*. Retrieved maart 2012, from codebetter.com: <http://codebetter.com/gregyoung/2010/02/13/cqrs-and-event-sourcing/>

Young, G. (2010, februari 16). *CQRS, Task Based UIs, Event Sourcing agh!* Retrieved maart 2012, from codebetter.com: <http://codebetter.com/gregyoung/2010/02/16/cqrs-task-based-uis-event-sourcing-agh/>

Young, G. (2010, februari 20). *Why use Event Sourcing?* Retrieved maart 2012, from codebetter.com: <http://codebetter.com/gregyoung/2010/02/20/why-use-event-sourcing/>

Bijlage C: Plan van Aanpak

Bachelorproject 2012

Voor de Bachelor Technische Informatica, TU Delft

Maarten van Meijeren, Yorick Holkamp

In samenwerking met *E.Novation*

Begeleiders

Bastiaan Bakker (E.Novation, afdeling Application Development)

Jan Hidders (TU Delft, afdeling Web Information Systems)



Inhoud

1.0 Inleiding.....	3
1.1 Plan van aanpak	3
1.2. Schets van het bedrijf	3
1.3. Achtergrond en aanleiding van de opdracht	3
2.0 Opdrachtoomschrijving.....	4
2.1 Inleiding.....	4
2.2 Opdrachtgever	4
2.3 Contactpersonen.....	4
2.4 Probleemstelling	4
2.5 Doelstelling	5
2.6 Opdrachtformulering.....	5
2.7 Op te leveren producten.....	5
2.8. Randvoorwaarden.....	5
2.9. Risicofactoren	6
3.0 Aanpak	7
3.1. Inleiding.....	7
3.2 Methodiek.....	7
3.2.1 Ontwikkelingsmethodiek Scrum	7
3.2.2 Vergelijking van ontwikkeltijd.....	7
3.3 Technieken	8
3.4 Werkzaamheden	8
3.5 Planning.....	8
4.0 Projectinrichting.....	9
4.1 Inleiding.....	9
4.2 Betrokkenen.....	9
4.3 Informatie	9
4.4 Faciliteiten.....	9
4.5 Uitgangspunten.....	9
5 Kwaliteitsborging	10
5.1 Inleiding.....	10
5.2 De kwaliteit	10
5.2.1. Documentatie	10
5.2.2. Versiebeheer	10
5.2.3 Evaluatie.....	10
Bijlage A: Planning.....	11

1.0 Inleiding

1.1 Plan van aanpak

In dit document zal uiteen gezet worden hoe wij verwachten het bachelorproject aan te pakken. Het project vindt plaats bij E.Novation te Rivium in Capelle aan den IJssel. Dit document zal ingaan op de achtergrond van het bedrijf om enige context te geven, dat is van toepassing bij dit verslag en komende verslagen. De opdrachtschrijving en -scope zullen hierna aan bod komen gevolgd door meer inhoudelijke zaken als de aanpak, projectinrichting en kwaliteitsborging van het project.

1.2. Schets van het bedrijf

E.Novation is een Europees / internationaal ICT-bedrijf dat verschillende IT-diensten verleent, zowel maatwerk als off-the-shelf oplossingen en business consulting. Het bedrijf is hiervoor onderverdeeld in specifieke business units met elk een eigen taak. De afdeling binnen het bedrijf waar deze opdracht zich afspeelt is de afdeling Managed Services, deze biedt een set diensten aan volgens het Software-as-a-Service-principe waarbij de software wordt ondergebracht bij de onsite datacenters.

Binnen deze set diensten speelt de E.Berichtendienst een grote rol, hiermee worden verschillende beveiligde communicatiesystemen aangeboden zoals ENS, welke onder andere gebruikt wordt door TNT Post, scheepvaart-communicatiedienstverlener Portbase en China Shipping.

1.3. Achtergrond en aanleiding van de opdracht

In het algemeen werkt E.Novation hierbij voornamelijk voor de overheid en semioverheid. Inmiddels is er een nieuw project 'from scratch' bezig om zorgverleners en verzekeraars onderling te laten communiceren over verleende diensten. Hierbij is gekozen om op basis van de Command Query Responsibility Segregation (CQRS) en Event Sourcing patterns te gaan ontwikkelen.

In gesprek met het bedrijf is naar boven gekomen is dat het wellicht voordelen zou kunnen bieden op meerdere vlakken wanneer dit gecombineerd wordt met een NoSQL-database als Event Store of query database. Uiteindelijk is hierbij besloten dit uit te laten zoeken door studenten in het kader van een bachelorproject.

2.0 Opdrachtomschrijving

2.1 Inleiding

In de deze sectie zullen meer details over de regelingen omtrent de opdracht toegelicht worden. Hierbij komt onder meer aan bod meer met wie wij als studenten te maken zullen hebben, wat de precieze doelstelling van het project is en welke onderzoeksvragen geformuleerd zijn. Daaruit volgen de op te leveren producten. Als slot informatie over welke randvoorwaarden er aan het project zijn verbonden en welke eventuele risico's aan dit project zitten.

2.2 Opdrachtgever

E.Novation is de opdrachtgever, binnen het bedrijf treedt Bastiaan Bakker op als personificatie van de opdrachtgever in kader van het bachelorproject.

2.3 Contactpersonen

Contactpersoon bij de TU Delft is dr.ir. A.J.H. Hidders van TU Delft, A.J.H.Hidders@tudelft.nl.

Contactpersoon bij E.Novation is Bastiaan Bakker, bastiaan.bakker@novation.nl.

2.4 Probleemstelling

Onze probleemstelling is: "Hoe kan er binnen E.Novation gewerkt worden met NoSQL databases binnen applicaties ontwikkeld met de CQRS en ES design patterns?"

Om deze probleemstelling te kunnen beantwoorden zullen wij proberen de volgende deelvragen te beantwoorden:

1. Welke NoSQL-database sluit het beste aan op de interne en externe kwaliteitseisen? (Meer over kwaliteitseisen, zie hoofdstuk 5)
2. Welke mogelijkheden en limitaties geeft een NoSQL database ten opzichte van de huidige SQL-oplossing?
 - a. Welke voordelen geeft de toepassing van NoSQL als Event Store binnen ES?
 - b. Welke voordelen geeft de toepassing van NoSQL als view database?
 - c. Welke toepassingen profiteren weinig of niet van een NoSQL-oplossing?
3. Wat zijn de gevolgen voor het ontwikkelproces van E.Novation?
 - a. Moet het ontwikkelproces gewijzigd worden?
 - b. Hoe kunnen systeemwijzigingen op later moment aan doorgevoerd worden?
 - c. In hoeverre moeten de applicatiebeheerders opgeleid worden?
 - d. Verhoogt het gebruik van CQRS/ES de efficiëntie van de ontwikkeling?
4. Wat zijn de performance-grenzen wanneer er gewerkt wordt met een NoSQL database?
 - a. Hoe ver reikt de performance van een enkele server?
 - b. Hoeveel winst levert het toevoegen van extra servers aan de database cluster?

2.5 Doelstelling

E.Novation wil kennis opdoen omtrent het gebruik van NoSQL-databases om zo bij toekomstige projecten een geïnformeerde beslissing te maken om deze technologie al dan niet toe te passen. Momenteel is er nog geen keuze gemaakt voor een specifiek NoSQL databasesysteem. Om tot een keuze te komen zullen wij een literatuuronderzoek uitvoeren en de beschikbare systemen toetsen aan de hand van de benodigde functionaliteit en gehanteerde kwaliteitseisen van het bedrijf.

Nadat bekend is welke DBMS het best aansluit op de kwaliteitseisen van E.Novation zal er een demo-project gemaakt worden om ontwikkelaars kennis met deze toepassing te laten maken met de toepassing van NoSQL in combinatie met CQRS en Event Sourcing. Hierbij kan er tevens een 'sandbox' aangeboden worden voor eventuele toekomstige proof of concepts.

2.6 Opdrachtformulering

Het onderzoeken van de toepassingsmogelijkheden van NoSQL databases binnen de huidige set van SaaS-diensten in het portfolio van E.Novation. In de oriëntatiefase worden de beschikbare NoSQL oplossingen aan de hand van een literatuurstudie vergeleken op basis van relevante functionaliteit, externe kwaliteitsattributen zoals robuustheid, prestaties, schaalbaarheid en interne kwaliteitsattributen zoals analyseerbaarheid, aanpasbaarheid, testbaarheid.

Vervolgens wordt een proof-of-concept uitgewerkt op basis van een van de bestaande toepassingen van E.Novation of een representatief fictief voorbeeld, waarbij gebruik gemaakt wordt van het Command Query Segregation (CQRS) pattern in combinatie met Event Sourcing. Hieraan zullen diverse tests worden onderworpen om bovengenoemde kwaliteitsattributen in de praktijk te meten. Daarnaast zal een statische kwaliteitsanalyse plaatsvinden (eventueel deels via de Software Improvement Group).

2.7 Op te leveren producten

Aan het einde van het project zullen twee applicaties opgeleverd worden, hierbij zal een gebruik maken van een traditionele relationele database en de andere applicatie van CQRS, Event Sourcing en een NoSQL database. Deze applicaties kunnen 'from scratch' gemaakt worden of mogelijk een bestaande open-source demo-applicatie zijn die verder uitgebreid wordt, hierbij kan bijvoorbeeld gedacht worden aan de 'Pet Store'-voorbeeldapplicatie van Sun.

Bij deze applicaties zal documentatie geleverd worden, zowel documentatie omtrent de applicaties zelf wat betreft het gebruik en de werking van de source code als documentatie over het proces rond de ontwikkeling van de applicaties. Hierbij zal er ook gekeken worden naar verschillen in ontwikkeltijd en onderhoudbaarheid tussen de twee systemen.

2.8. Randvoorwaarden

De studenten werken minimaal 24 uur per week en zijn op gelijke dagen aanwezig. Aanvang tussen 8:30 en 10:30, eindtijd tussen 17:00 en 19:00. Daarnaast is vijftig dagen het minimale totaal dat de studenten aan het project werken.

2.9. Risicofactoren

Binnen nieuwe projecten zijn er in alle gevallen risicofactoren aan te wijzen, dit project is hier uiteraard geen uitzondering op. Het doel van dit project is, zoals in de doelstelling tevens zichtbaar is, het werken met NoSQL in combinatie met CQRS/ES, iets waar binnen het bedrijf geen ervaring mee is. Wel is er een mogelijkheid tot advies vanuit de TU Delft, Jan Hidders is binnen EWI de expert op het gebied van databases.

Naast dit inhoudelijke punt vormt ook de focus van het project een risico, momenteel zijn nog niet alle zaken rondom de opdracht volledig afgedekt wat betekent dat wij deels verantwoordelijk zijn voor het vastleggen van een focus die goed bij de probleem- en doelstellingen past. Bij op dit punt komt ook een spanningsveld aan het licht; het bedrijf is op zoek naar ervaring met deze combinatie van technieken en de vergelijking tussen de nieuwe en 'klassieke' aanpak terwijl er voor de TU Delft een voldoende diepgaand softwareproject neergezet moet worden, twee punten die niet noodzakelijk op één lijn liggen.

Scrum vormt een andere risicofactor binnen het project, specifiek voor ons als studenten. Wij hebben geen ervaring met het werken via Scrum. Daarnaast is het bij Scrum van belang een goede backlog op te stellen en daarbij in te schatten hoeveel tijd per story benodigd zal zijn. Omdat het een nieuw systeem betreft wat 'from scratch' opgebouwd gaat worden kan dit een risico vormen voor de haalbaarheid van sprints.

Tot slot zullen we ook tegen het probleem aan blijven lopen dat we voor bepaalde bedrijfsspecifieke informatie afhankelijk zijn van bepaalde medewerkers, die niet op elk moment aanwezig zijn en hun eigen werkzaamheden hebben. Daarnaast is onze begeleider tevens druk bezet waardoor er voldoende ruimte ingepland zal moeten worden informatie uit te wisselen. Deze twee factoren zullen betekenen dat er in sommige gevallen gewerkt zal moeten worden met de beperkte informatie die op dat moment beschikbaar is wat voor tijdverlies zou kunnen zorgen.

Om het project in goede banen te leiden zullen we dus goed op de voortgang moeten blijven letten, voldoende tijd voor informatievergaring moeten inplannen en bovenal snel focus aan moeten brengen binnen de opdracht. Hiermee kan het grootste struikelblok, namelijk een tijdgebrek, vermeden worden wat ons in staat zou moeten stellen op tijd een product op de kunnen leveren dat voldoet aan de gestelde eisen.

3.0 Aanpak

3.1. Inleiding

In de aanpak zullen de methoden en technieken besproken worden die gebruikt gaan worden in het project. Daarnaast zal er een blik geworpen worden op de werkzaamheden die wij uit gaan voeren. Afgesloten met een vooruitzicht van de grove tijdsindeling van de werkzaamheden.

3.2 Methodiek

3.2.1 Ontwikkelingsmethodiek Scrum

De Scrum-ontwikkelmethode is de standaard voor ontwikkeling binnen E.Novation, deze methode valt onder de noemer van Agile programming. Dat is een principe van iteratief ontwikkelen, elke iteratie wordt een bepaald gedeelte van de applicatie ontwikkeld. Scrum is een proces waarin requirements kunnen 'evalueren' en snel kan worden ingegaan op veranderingen. Voor meer details over Scrum verwijzen wij naar het Oriëntatieverslag hoofdstuk negen.

Eventueel wordt er gebruik gemaakt van Pair Programming waarin twee ontwikkelaars samen achter één PC programmeren. De ene ontwikkelaar programmeert terwijl de andere de code beoordeelt, over de methoden nadenkt en eventueel randzaken opzoekt. Zo wordt degene die de code typt ontlast en kan zich volledig concentreren op het programmeren.

3.2.2 Vergelijking van ontwikkeltijd

Om de efficiëntie van ontwikkeling te vergelijken tussen de twee te gebruiken methoden zal gekeken worden naar de benodigde ontwikkeltijd. Een voorwaarde om deze vergelijking goed te kunnen maken is dat de vergeleken systemen zover mogelijk aan elkaar gelijk zijn waarbij enkel de zaken die uniek zijn voor elk van de twee systemen afwijken om op deze manier zoveel mogelijk 'ruis' te voorkomen.

De vergelijking kan in twee stukken worden verdeeld, namelijk de ontwikkelingstijd van nieuwe features en de tijd om huidige functionaliteit aan te passen. Hierbij kan bij nieuwe functionaliteit gedacht worden aan het toevoegen van ondersteuning voor afdelingen waar contacten in een contactenlijst toe kunnen behoren, het aanpassen van bestaande functionaliteit zou dan kunnen zijn het toevoegen van nieuwe constraints op de in te voeren data.

In de vergelijking moet daarnaast rekening gehouden worden met tal van factoren die invloed hebben op de ontwikkeltijd, zoals de vaardigheden van de programmeur en de condities van de werkomgeving. Er worden dan zoveel mogelijk factoren buitengesloten op het verschil van de systemen na. De vergelijking in tijd heeft dan de grootst mogelijke betrekking op het verschil van de systemen. Om zoveel mogelijk op één lijn te trekken worden de systemen uitgebreid met dezelfde functionaliteit en door dezelfde programmeur.

Daarnaast kan de tijdvergelijking worden gebruikt om veranderingen in het systeem te meten, het herschrijven van bepaalde methoden etc. Dan wordt er geen nieuwe functionaliteit toegevoegd, maar wordt het systeem op een bepaalde manier onderhouden. Dat is uiteraard ook van belang.

Het uitbreiden van functionaliteit is onder te verdelen in het inlezen en begrijpen van de systemen en het daadwerkelijk schrijven en testen van de functies. Doordat de functie altijd twee maal wordt geschreven zijn de functionaliteit en tests al een keer geïmplementeerd wat invloed zal hebben op de tweede keer dat dezelfde functie wordt geschreven. Een manier om dit zo neutraal mogelijk te doen is om de systemen die eerst geïmplementeerd worden zoveel mogelijk te wisselen en mogelijk het schrijven van de (voor zover mogelijk) generieke delen, zoals bijvoorbeeld test cases op een hoger niveau, buiten beschouwing te laten.

3.3 Technieken

De volgende technieken zullen gebruikt gaan worden in het project, Java, NoSQL, CQRS en Event Sourcing. Java is de standaard programmeertaal binnen E.Novation. Het onderzoek zal bekijken wat de toegevoegde waarde van NoSQL zou kunnen zijn. Dit is wenselijk om in een zo realistisch mogelijke omgeving te laten plaatsvinden, zodoende zullen wij Java gebruiken.

Voor de opslag van gegevens zal gekeken worden naar een NoSQL databasesysteem, dit is een non-relationale databasevorm die nog niet wordt toegepast binnen E.Novation.

Naast NoSQL zijn er nog twee principes waar wij gebruik van gaan maken. CQRS en Event Sourcing, CQRS houdt grofweg in dat zoek en schrijf methoden binnen de code gescheiden worden. Event Sourcing bewaart in plaats van de uiteindelijke objecten de serie van events die leiden tot de uiteindelijke staat van het object. Deze kan in de applicatie met behulp van de events gereconstrueerd worden.

In het Oriëntatieverslag wordt uitgebreider ingegaan op de gebruikte technieken, voor meer informatie verwijzen wij dan ook naar hoofdstukken zes tot en met acht.

3.4 Werkzaamheden

De werkzaamheden zullen bestaan uit het onderzoeken van NoSQL databases, het vormen van een gefundeerde keuze voor een NoSQL database die aansluit op de nader te onderzoeken eisen van E.Novation. Daarnaast het bouwen van een applicatie die als demoproject kan fungeren, met de eigenschappen NoSQL, CQRS en Event Sourcing.

3.5 Planning

Doordat er tijdens het project volgens de Scrum-methodiek is gewerkt bevat onze planning enkel een globale verdeling van het project in de tijd. Afhankelijk van de prioriteiten van de Product Owner zal er beslist worden welke functionaliteit van het systeem op welk moment ontwikkeld zullen worden.

De globale planning is bijgevoegd als Bijlage A.

4.0 Projectinrichting

4.1 Inleiding

De projectinrichting omvat details over wie er met het project te maken hebben, hoe de algemene voortgang wordt geëvalueerd en welke faciliteiten er tot de beschikking worden gesteld. Daarnaast wordt er kleine toelichting gegeven over de uitgangspunten van de te vergelijken projecten.

4.2 Betrokkenen

Binnen E.Novation zal de dagelijkse gang van zaken rond het project begeleid worden door Bastiaan Bakker, teamleider IT, die tevens als Product Owner en Scrum Master op zal treden. Tevens is er periodiek contact met Dré Lameir, quality manager, met wie algemene zaken over het project en de werkzaamheden besproken zullen worden. Vanuit de TU Delft is Jan Hidders de begeleider, op hem kunnen we terugvallen als er inhoudelijke problemen zijn waar geen ervaring vanuit E.Novation mee is.

4.3 Informatie

Communicatie en rapportage over het project zal dagelijks tijdens de daily standup meeting met de teamleden afgehandeld worden. Aan het eind van iedere sprint vindt er een evaluatie van de afgelopen sprint plaats, de sprint retrospective, waarin deze informatiestroom samen met de algemene voortgang geëvalueerd zal worden.

4.4 Faciliteiten

Er wordt tijdens het project gebruik gemaakt van het kantoor van E.Novation, waarbij er een workstation met twee workplaces beschikbaar gesteld zal worden voor ontwikkeling en overige werkzaamheden.

4.5 Uitgangspunten

Als uitgangspunt van de vergelijking worden twee verschillende projecten genomen. De projecten moeten qua functionaliteit redelijk overeenkomen om tot een zinnige vergelijking te kunnen komen. De projecten die in dit onderzoek gebruikt gaan worden dienen hetzelfde doel, namelijk het zijn alle twee adresboeken. Beide zijn bestaande open-source applicaties. Een van de twee is gebaseerd op het Axon Framework¹, het andere project is een 'reguliere' JPA applicatie². (De verwijzing naar de bron van de code is toegevoegd in de voetnoot.) Het "Axon Framework"-project voldoet standaard nog niet aan de eisen die gesteld zijn, dit project werkt nog met een reguliere SQL-database terwijl het onderzoek draait om een NoSQL database in combinatie met CQRS/ES. Deze SQL database wordt verwijderd en daarvoor in de plaats komt een NoSQL database.

Om een eerlijke vergelijking te kunnen maken moeten de projecten zo veel mogelijk op elkaar lijken. Dat gaat natuurlijk vooral over de geboden functionaliteit. De functionaliteit in beide projecten moet zo gelijk mogelijk zijn wat nu niet het geval is. Het is de bedoeling om eerst de twee projecten gelijkwaardig te maken qua functionaliteit. Als dat is gebeurd kunnen beide uitgebreid worden met dezelfde features, daarop worden verschillende vergelijkingen gedaan.

¹ <https://github.com/AxonFramework/Addressbook-Sample>

² <https://code.google.com/p/mediapool/source/browse/trunk/source/jpacontainer-addressbook-demo/?r=121>

5 Kwaliteitsborging

5.1 Inleiding

Kwaliteit is belangrijk, om voor goede kwaliteit software te zorgen zullen wij meerdere methoden om de kwaliteit te waarborgen gebruiken. Een van deze punten is het bijhouden van accurate documentatie, zowel van het systeem als van problemen die tijdens de ontwikkeling van de software tegenkomen. Vervolgens zal besproken worden hoe gewerkt zal worden met versiebeheer, op welke manier de kwaliteit geëvalueerd zal worden en tot slot hoe de software mogelijk tot pilot verheven zou kunnen worden.

5.2 De kwaliteit

5.2.1. Documentatie

Een van de speerpunten van een hoogstaand softwareproject is de aanwezigheid van kwalitatief hoogstaande en volledige documentatie. Om hier tijdens ons project zo goed mogelijk op in te spelen zal zowel het proces als de software zelf gedocumenteerd worden.

De documentatie van het proces heeft als doel de tegengekomen problemen tijdens de ontwikkeling vast te leggen en te delen met de ontwikkelaars. Op deze manier wordt dit project een pilot wat betreft de gecombineerde technieken, door vervolgens fouten vast te leggen kunnen mogelijke toekomstige projecten op basis van dezelfde set technieken voorspoediger verlopen.

Het documenteren van de software zelf heeft vanzelfsprekend als doel de onderhoudbaarheid en bruikbaarheid van de applicatie te verhogen. De broncode zal voorzien worden van documentatie met behulp van Javadoc om de comments en uitleg binnen de broncode ook buiten de ontwikkelomgeving beschikbaar te maken en toekomstige ontwikkelaars of gebruikers handvatten te bieden om het systeem snel onder de knie te kunnen krijgen.

5.2.2. Versiebeheer

Binnen het bedrijf wordt gewerkt met het Subversion versiebeheersysteem. Dit systeem kan veranderingen aan de software bijhouden en delen vanuit een centrale repository, de plek waar alle bestanden worden bijgehouden. Om het overzicht te behouden wordt hierbij per Scrum-story een nieuwe branch aangemaakt. Hiermee zijn wijzigingen per story duidelijk te onderscheiden. Dat heeft als voordeel dat eventuele fouten snel opgespoord en verholpen kunnen worden.

5.2.3 Evaluatie

Binnen het bedrijf is het gangbaar alle projecten te koppelen aan een Jenkins³ 'continuous integration server' om zo periodiek de projecten te compileren en bij te houden of hier fouten bij ontstaan.

Vanuit Jenkins wordt er gewerkt met het opensource project Sonar⁴ om de softwarekwaliteit te monitoren. Sonar houdt een reeks aan code metrics bij, waaronder de hoeveelheid comments, code duplication, hoeveelheid lines of code, cyclomatic complexity per methode en verschillende andere zaken. Hiermee kan met één set regels een hele collectie aan applicaties op gelijke wijze doorgelicht worden en op duidelijke wijze verbeterpunten inzichtelijk gemaakt worden.

³ <http://jenkins-ci.org/>

⁴ <http://www.sonarsource.org/>

Bijlage A: Planning

[illegible]

Bijlage D: Research document

Bachelorproject 2012

Voor de Bachelor Technische Informatica, TU Delft

Maarten van Meijeren, Yorick Holkamp

In samenwerking met *E.Novation*

Begeleiders

Bastiaan Bakker (E.Novation, afdeling Application Development)

Jan Hidders (TU Delft, afdeling Web Information Systems)



Samenvatting

In opdracht van E.Novation is er onderzoek gedaan om een NoSQL-database te vinden die aansluit op de gestelde kwaliteitseisen van het bedrijf. Deze eisen zijn opgesteld aan de hand van gesprekken met de verschillende actoren binnen het bedrijf. Uit deze gesprekken zijn primaire en secundaire eisen voortgekomen, hierbij wordt onderscheid gemaakt tussen de zaken waaraan voldaan moet worden en de zaken die als pré gelden.

Op basis van deze eisen is een lijst van NoSQL-databases doorlopen en is er een voorselectie gemaakt van relevante databases op basis van de gestelde primaire eisen. Hierna zijn de databases HBase, Cassandra, MongoDB en CouchDB verder behandeld in het onderzoek.

In het verdere onderzoek met betrekking tot de kwaliteitseisen zijn de specificaties en documentatie van deze vier systemen getoetst aan de eisen waarbij de primaire kwaliteitseisen zwaarder meewegen dan de secundaire eisen. Uit dit onderzoek is gebleken dat MongoDB op de gegeven punten beter dan of gelijkwaardig aan de concurrentie scoort zoals tevens zichtbaar is in Tabel 1 en dit systeem zal voorgedragen worden om in de vervolgfase van dit project te gebruiken.

	MongoDB	CouchDB	HBase	Cassandra
Primaire kwaliteitseisen				
Functionaliteit:	++	+	+	+
Betrouwbaarheid:	++	+	++	++
Onderhoudbaarheid:	++/+	+/-	+	++/+
Secundaire kwaliteitseisen				
Bruikbaarheid:	++	++/+	+	+
Vervangbaarheid:	-	-	-	-
Efficiëntie:	+	+	+	+
Eindoordeel:	++/+	+	+	+

Tabel 1 de resultaten van het onderzoek

Inhoud

Samenvatting	2
1. Inleiding.....	4
2. Kwaliteitsattributen	5
2.1 Primaire eisen	5
2.2 Secundaire eisen	5
3. Voorselectie databasesystemen	6
3.1 Key-value, XML en Object stores	6
3.2 Document stores.....	6
3.3 Wide column stores	7
3.4 Selectie.....	7
4. Onderzochte databasetypen.....	7
4.1 Document stores.....	7
4.1.1 Datamodel.....	7
4.1.2 Functionaliteit	8
4.1.3 Betrouwbaarheid	8
4.1.4 Robuustheid.....	9
4.1.5 Ondersteuning	9
4.1.6 Schaalbaarheid.....	10
4.2 Wide column stores	10
4.2.1 Datamodel.....	10
4.2.2 Functionaliteit	11
4.2.3 Betrouwbaarheid	12
4.2.4 Robuustheid.....	12
4.2.5 Ondersteuning	12
4.2.6 Schaalbaarheid.....	12
5. Afweging	14
5.1 Functionaliteit	14
5.2 Betrouwbaarheid	14
5.3 Onderhoudbaarheid.....	15
5.3.1 Licentie	15
5.3.2 Aanpasbaarheid	15
5.3.3 Server monitoring	15
5.3.4 Stabiliteit	16
5.4 Bruikbaarheid.....	17
5.4.1 Duidelijkheid concept	17
5.4.2 Documentatie	17
5.5 Vervangbaarheid.....	18
5.6 Efficiëntie	18
5.6.1 Middelenbeslag.....	18
5.6.2 Schaalbaarheid.....	20
6. Conclusie	21
Bijlage A: Onderzochte systemen	22
Bijlage B: Woordenlijst.....	23

1. Inleiding

Dit document bevat het onderzoek naar welke database het beste voldoet aan de in het oriëntatieverslag genoemde kwaliteitseisen, alle geselecteerde databases zijn in bijlage A te vinden. Woorden die uitleg behoeven zijn te vinden in bijlage B, de woordenlijst. Om te beginnen zijn de eisen waaraan de databases moeten voldoen opgesomd. Gevolgd door de voorselectie van de databases, op basis van de daarvoor genoemde primaire kwaliteitseisen. Met deze voorselectie willen we de focus zoveel mogelijk vernauwen. De reden daarvoor is dat alleen onderzoek gedaan moet worden naar de databases die echt relevant zijn voor dit onderzoek. De databases die overblijven na de voorselectie worden onderworpen aan een uitgebreider onderzoek. Deze worden vergeleken op basis van de primaire en secundaire kwaliteitsattributen. Als het onderzoek naar de eigenschappen is afgerond kunnen we een keuze maken op basis van deze kwaliteitsattributen. De database die als beste uit de test naar voren komt zal gebruikt worden in het vervolg van dit project.

2. Kwaliteitsattributen

In de oriëntatiefase hebben wij verschillende stakeholders gesproken, op basis van deze gesprekken zijn we op verschillende kwaliteitsattributen uitgekomen waar de database aan moet voldoen. De eisen zijn onderverdeeld in primaire en secundaire eisen, de primaire eisen zijn de belangrijkste eisen aan de database.

2.1 Primaire eisen

- Functionaliteit
 - Te gebruiken met een applicatie geschreven in Java.
 - Beschikt over voldoende functionaliteit om in te passen in toepassingen binnen E.Novation.
- Betrouwbaarheid
 - Foutbestendig bij uitval van servers.
 - Herstelbaar na fouten en uitval van servers.
- Onderhoudbaarheid van de databaseserver
 - Mogelijkheid tot het analyseren van het gedrag van de server, een openbare broncode is hierbij een pre om gedrag te kunnen verklaren.
 - Mogelijkheid tot het aanpassen van de software en deze wijzigingen kunnen testen op neveneffecten in de overige code.
 - Beschikbaarheid van tooling om de status van servers bij te houden.
 - Voldoende stabiel voor een productieomgeving, hierbij gelet op zowel eventuele grote wijzigingen in de API, kritieke bugs en voldoende toekomstperspectief.

2.2 Secundaire eisen

- Bruikbaarheid
 - Voldoende begrijpelijke concepten voor ontwikkelaars.
 - De leerbaarheid van de systemen vormt geen grote drempel.
- Vervangbaarheid
 - De mogelijkheid om op enig moment zonder vergaande applicatiewijzigingen te kiezen voor een alternatief databasesysteem.
- Efficiëntie
 - Geen buitensporig middelenbeslag ('resource utilisation').
 - Schaalbaar om in mogelijke toekomstige groei te kunnen voorzien.

3. Voorselectie databasesystemen

In de gesprekken met verschillende stakeholders binnen het bedrijf zijn de kwaliteitsattributen naar voren gekomen die in hoofdstuk 2 genoemd zijn. Op basis hiervan hebben wij een voorselectie gemaakt van databasesystemen die volledig of grotendeels aan de primaire kwaliteitsattributen voldoen. De totale selectie van onderzochte databases staat in bijlage A deze databases komen van onder andere de nosq-databases.org¹ site.

Enkele punten die naar voren gekomen zijn in de kwaliteitseisen zijn doorzoekbaarheid van de opgeslagen data, op zowel attributen als tekstuele inhoud, daarnaast flexibiliteit van het databaseschema. Dit zijn twee belangrijke features voor de beoogde database. Bovendien moet de mogelijkheid bestaan een dataset te behandelen die groter is dan de hoeveelheid geheugen in een individuele machine.

3.1 Key-value, XML en Object stores

Op basis van de voorgaande primaire eisen vallen alle door ons bekeken key-value stores af. Omdat ofwel de data niet op intelligente (MapReduce) wijze te doorzoeken is, (Membase, Kyoto/Tokyo Cabinet) en/of er niet gewerkt kan worden met een dataset die groter is dan de hoeveelheid werkgeheugen die in de machine aanwezig is (Redis).

Naast de key-value stores vallen de XML-stores ook in de categorie die niet snel doorzoekbaar is. In een normale database wordt alleen het veld uitgelezen waarin er gezocht moet worden. Bij XML worden de tags en andere informatie ook uitgelezen. Dit maakt het een zoeken een stuk inefficiënter. Omdat zoeken een belangrijk aspect is vallen ook de databases gebaseerd op XML af. Object stores zijn over het algemeen matig doorzoekbaar en kunnen geen batchupdates afhandelen, Object store databases blijken ook geen goede kandidaat te zijn.

3.2 Document stores

Onder document stores zijn er ook meerdere opties die wegvallen doordat ze aan één of meerdere eisen niet kunnen voldoen. Daaronder vallen de volgende databases: ThruDB, RavenDB, Persevere, SisoDB, Terrastore, Riak, Hypertable, Citrusleaf en Clusterpoint Server. Deze databases zullen we dan ook niet verder uitwerken. Hieronder een korte toelichting waarom de databases niet aan de eisen voldoen.

ThruDB is niet langer in ontwikkeling en valt hiermee af. Bij anderen ontbreekt een goede adapter naar Java, de databases RavenDB, SisoDB en Perservere hebben daar last van. Er is wel een omslachtige manier om deze aan Java te koppelen, echter gaat hierbij performance verloren en wordt de complexiteit groter. Terrastore, Riak, SisoDb, Persevere en Hypertable hebben een erg beperkte community en/of weinig releases wat ze in ieder geval niet tot de eerste keuze maakt. Citrusleaf heeft net als Clusterpoint Server commerciële ondersteuning, tegen betaling kan er een license verkregen

¹ <http://nosql-database.org/>

worden waarbij de code niet open source zal zijn, dit is niet wenselijk. Bij Clusterpoint en Jackrabbit is er geen MapReduce beschikbaar, deze databases zijn dus niet op de gewenste manier te doorzoeken.

Geen Java-support	Geen MapReduce en multiple indexes	Onvoldoende actieve community
Persevere RavenDB SisoDB	Clusterpoint Server Jackrabbit SisoDB	Citrusleaf Terrastore ThruDB

Tabel 2: bronnen van de databases in Bijlage A

3.3 Wide column stores

In de voorselectie van de wide column stores valt Amazon SimpleDB af. Deze database kan alleen bij Amazon opgeslagen worden, dat valt af aangezien de database op de in-house servers van E.Novation dient te draaien. Hypertable, tevens een wide column store, valt af door de beperkte community van dit relatief nieuwe project.

3.4 Selectie

Op basis van deze voorselectie zijn MongoDB, CouchDB als document stores nog mogelijke kandidaten. De wide column stores die overblijven zijn Cassandra en HBase. Alle key-value stores zijn om de eerder beschreven redenen niet geschikt bevonden.

4. Onderzochte databasetypen

In ons onderzoek hebben wij uiteindelijk een tweetal NoSQL-databasetypen onderzocht, namelijk document stores en wide column stores. Deze twee typen systemen hebben een dusdanig verschillende aanpak dat deze los van elkaar toegelicht zullen worden. In het volgende hoofdstuk worden eerst de document stores uitgewerkt, gevolgd door de wide column stores.

4.1 Document stores

4.1.1 Datamodel

In de bekeken document stores wordt data gerepresenteerd in de vorm van een map in JSON (JavaScript Object Notation). De opslag hiervan gebeurt voor CouchDB op identieke wijze. In het geval van MongoDB² wordt de data in BSON (Binary JSON)³ behandeld, een binaire variant op JSON. De documenten van MongoDB zijn gelimiteerd tot een grootte van 16MB met de filosofie dat wanneer een enkel document groter wordt er waarschijnlijk een betere verdeling van de data mogelijk zal zijn. CouchDB⁴ stelt heeft een limiet van 1MB⁵.

De documenten kunnen verschillende typen data bevatten waaronder de gangbare primitieve typen als integers, doubles, binaire data en strings, daarnaast is er tevens ondersteuning voor meer complexe

² <http://www.mongodb.org/display/DOCS/Schema+Design>

³ <http://bsonspec.org/>

⁴ <http://wiki.apache.org/couchdb/Technical%20Overview>.

⁵ <http://stackoverflow.com/questions/28975/anyone-using-couchdb/118273#118273>

datatypen als arrays en maps. Hierbij kunnen arrays en maps tevens deze typen bevatten waardoor er een geneste structuur kan ontstaan.

Om structuur aan te brengen in de verzamelingen documenten worden er verschillende methoden gebruikt door de twee kandidaten. MongoDB brengt structuur aan door de documenten onder te brengen in 'collections', vergelijkbaar met een tabel in een RDBMS, waarbij het aan de gebruiker is om te kiezen welke documenten in welke collectie behoren. De database kan vervolgens per collectie doorzocht worden op documenten.

CouchDB heeft een andere manier om dit probleem aan te pakken, hierbij worden documenten in één grote verzameling opgeslagen en kan er een selectie van de documenten bekeken worden door middel van vooraf gedefinieerde views. Een view is een JavaScript-functie die een document als argument ontvangt en deze afhankelijk van een berekening of niets of een document met te gebruiken index retourneert, gelijk aan de map-functie van het MapReduce-principe.

4.1.2 Functionaliteit

De databaseservers communiceren met applicaties over sockets (MongoDB) of HTTP-API (CouchDB), hiervoor zijn ondersteunde drivers voor beschikbaar in de meeste gangbare programmeertalen als C, C++, Java, Erlang en verschillende.NET-talen. De web-API van CouchDB is een REST-interface (REpresentational State Transfer)⁶ waarbij er voor MongoDB third party REST-interfaces beschikbaar zijn⁷ met gedeeltelijke functionaliteit. De MongoDB community is wel van plan om met een volledige versie te komen⁸.

Via de drivers is het bij MongoDB mogelijk om dynamische queries direct uit te voeren op de verschillende collecties van documenten met gebruik van reguliere expressies (dit geldt niet voor CouchDB), zelf gedefinieerde JavaScript-functies en logische voorwaarden, zowel over de directe waarden van het document als de mogelijke geneste waarden. Beide systemen hebben ondersteuning voor MapReduce-opdrachten die binnen de V8 JavaScript-engine uitgevoerd worden.

CouchDB heeft ook een Internet interface, Futon⁹. Dit is een web-based administration tool. Met deze tool is het mogelijk om individuele documents aan te passen, te verwijderen of nieuwe documenten toe te voegen. Er zit onder andere nog een replicator GUI en een test suite bij.

4.1.3 Betrouwbaarheid

Het verzorgen van betrouwbare dataopslag gebeurt over het algemeen door de data te repliceren op verschillende servers zodat er altijd een aantal servers offline kan gaan zonder dat er data verloren gaat of de applicatie wegvalt. Bij MongoDB¹⁰ gebeurt op basis van 'replica sets'¹¹, dit is een uitbreiding op de

⁶ <http://wiki.apache.org/couchdb/Basics>,

⁷ <http://www.mongodb.org/display/DOCS/Drivers>

⁸ <http://www.mongodb.org/display/DOCS/Comparing+Mongo+DB+and+Couch+DB>

⁹ http://wiki.apache.org/couchdb/Getting_started_with_Futon,

¹⁰ <http://www.mongodb.org/display/DOCS/Replication>

¹¹ <http://www.mongodb.org/display/DOCS/Replica+Sets+-+Basics>

klassieke master-slave replication. Een replica set is een collectie nodes welke een dataset delen (dit kan de gehele dataset zijn of dezelfde shards) en welke onderling een primary server kiezen wanneer de bestaande primary wegvalt. Schrijfacties naar de database worden aan de primary doorgegeven, welke deze wijzigingen vervolgens door de secondary servers overgenomen worden. Leesacties op de database kunnen door elke node afgehandeld worden.

Wanneer de primary een meerderheid van de secondary servers niet kan bereiken dan levert deze zijn primary status in. Zodra een meerderheid van de secondary servers een primary niet kan bereiken wordt er onderling een vervangende primary server gekozen. Mocht vervolgens de 'oude' primary weer online komen dan vervalt deze tot een secondary server en zal de server weer proberen te synchroniseren.

CouchDB¹² is gebaseerd op een peer-2-peer systeem van databases ook wel master-master replication genoemd. Op elke peer kan dezelfde data worden gewijzigd waarna later wordt besloten welke versie de uiteindelijke correcte versie is. De database kan ook lokaal worden binnengehaald, bijvoorbeeld op een laptop of een lokale server. Zo is het mogelijk om lokaal een gedeelte van de data die voor jouw applicatie relevant is te gebruiken en met de overige servers te synchroniseren op incrementele wijze.

4.1.4 Robuustheid

CouchDB werkt 'on disk' volgens het ACID principe, dat betekent dat de in het systeem verwerkte data te allen tijde consistent blijft. Om dit mogelijk te maken wordt gewerkt met een 'crash-only' principe waarbij zonder enige vorm van afsluitingsprocedure het systeem kan worden gestopt. Om dit mogelijk te maken worden de te ondernemen acties bij ontvangst meteen in de achtergrond naar disk geschreven. Zodra deze data naar de schijf geschreven wordt en het systeem crasht voordat alle acties verwerkt zijn dan worden de tot onvolledig weggeschreven delen als niet als behandeld beschouwd en worden de acties opnieuw uitgevoerd. Indien er een transactie plaatsvindt met meerdere acties dan blijven de headers bestaan om zo de acties op het juiste moment in het proces te hervatten zonder dat hierbij reparatiewerkzaamheden benodigd zijn.

MongoDB daarentegen heeft wel een reparatie nodig na een crash doordat, met de standaardinstellingen, niet meteen alle wijzigingen naar disk weggeschreven worden. MongoDB gebruikt wel vergelijkbaar met CouchDB een log met te nemen acties, bij MongoDB onder de noemer 'journaling', waar binnengekomen acties op weggeschreven worden. Wanneer de database vervolgens na een crash wordt opgestart speelt deze de log af vanaf het op dat moment bereikte punt en komt de database zo weer op de gewenste state uit.

4.1.5 Ondersteuning

MongoDB wordt ontwikkeld door 10gen¹³, een Amerikaans bedrijf dat tegen vergoeding support en trainingen aanbiedt. CouchDB is ondergebracht bij de Apache Foundation en wordt onderhouden door enkele bedrijven¹⁴. Naast de ontwikkelaars zijn er nog meerdere bedrijven die ondersteuning verlenen

¹² <http://guide.couchdb.org/draft/replication.html>

¹³ <http://www.mongodb.org/display/DOCS/Contributors>

¹⁴ http://wiki.apache.org/couchdb/Professional_Services

of MongoDB en CouchDB aanbieden volgens het SaaS-principe. Verder zijn er veel grote bedrijven te vinden die MongoDB inmiddels in productie hebben genomen.

4.1.6 Schaalbaarheid

De belangrijkste feature van MongoDB voor schaalbaarheid is sharding¹⁵, hierbij wordt een collectie van documenten op basis van een bepaalde key verdeeld over verschillende servers. Zodra vervolgens een query op een routing server binnenkomt wordt deze afhankelijk van de impact van de query naar de relevante shard (bij een zoek- of update-opdracht die binnen een of enkele shards valt) of alle servers verstuurd (zoals bij een zoekopdracht over de hele collectie).

Voordelen hiervan zijn dat verschillende operaties in parallel uitgevoerd kunnen worden op de dataset over verschillende servers. Dit kan verbeterde performance kan leveren bij bijvoorbeeld zoekacties en MapReduce. Hiernaast maakt sharding het mogelijk om een dataset te beheren die te groot is om op één server (efficiënt) te manipuleren doordat er slechts een deel van de dataset gehanteerd hoeft te worden. Hierdoor kunnen beperkingen op het gebied van disk space voorkomen worden of kan er gekozen worden voor het opdelen van de data zodat deze in meer gevallen binnen het werkgeheugen van de databaseserver past om zo hogere performance te kunnen leveren.

CouchDB standaard maakt geen gebruik van sharding om grote datasets mogelijk te maken, schalen gebeurt enkel doormiddel van replication, hierbij wordt de performance in het algemeen verhoogd maar zijn zeer grote datasets niet mogelijk. Wel zijn er meerdere ontwikkelaars die hier oplossing voor hebben gemaakt. Voor sharding in CouchDB zijn er 'BigCouch'¹⁶, 'Lounge'¹⁷, 'Gizzard'¹⁸ en 'Pillow'¹⁹. Al deze opties zijn open source en kunnen dienen als "sharding manager" voor CouchDB.

4.2 Wide column stores

Deze databases zijn gebaseerd op het concept uiteengezet door Google in een paper over het proprietaire BigTable databasesysteem²⁰. De principes waarop deze systemen zijn ontworpen verschillen sterk met de document stores wat zich met name uit in het datamodel, waar document stores een zeer flexibele en geneste structuur kennen hebben wide column stores in het algemeen een sterke mate van cohesie.

4.2.1 Datamodel

De Wide column stores werken op basis van het BigTable-principe, hierbij bevat elke row een vaste hoeveelheid ColumnFamily-kolommen, waarbij elke ColumnFamily een flexibel aantal kolommen kan

¹⁵ <http://www.mongodb.org/display/DOCS/Sharding>

¹⁶ <http://bigcouch.cloudant.com/>

¹⁷ <http://tilgovi.github.com/couchdb-lounge/>

¹⁸ <https://github.com/twitter/gizzard>

¹⁹ <https://github.com/khellan/Pillow>

²⁰ <http://research.google.com/archive/bigtable.html>

bevatten, voor Cassandra²¹ geldt een maximum van twee miljard. Daarnaast heeft Cassandra ook de mogelijkheid tot Super Column Families in plaats van Column Families. Een Super Column kan meerdere kolommen bevatten, dat betekent dat in figuur 1 de CF onderverdeeld kan zijn in nog een aantal kolommen. Equivalent aan het flexibel aantal kolommen in een Column Family kan de Super Column Family een flexibel aantal Super Columns bevatten. Uiteindelijk is behalve het aantal kolommen de hele database ook aanpasbaar, echter moet de database daar wel voor offline om wijzigingen toe te passen.

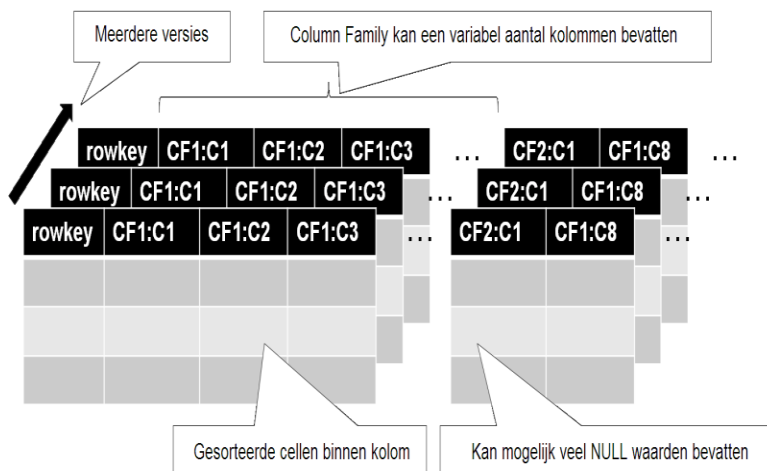
In HBase²² bevat elke row vervolgens per kolom een tot meerdere verschillende versies van de ingestelde waarde. Om een waarde in HBase op te vragen moet er dan ook door middel van het tuple (row, column, versie) in de database gezocht worden. Cassandra gebruikt een timestamp in plaats van de versie, dit is vooral bedoeld voor replication. Er wordt dan ook maar één versie van het bestand opgeslagen.

De daadwerkelijke data wordt opgeslagen als bytereeks. Hierdoor is het in principe mogelijk allerlei soorten data op te slaan. Als limitatie vermeldt de documentatie echter dat het 'niet verstandig' is

objecten van 10-50MB op te slaan, wat er op duidt dat de grootte per cell het beperkt gehouden moet worden. Er zijn verschillende beperkingen in Cassandra: Een column value mag niet groter dan 2GB zijn. Over het algemeen geldt dat kleine bestanden van een paar MB meer geschikt zijn, net als bij HBase. Ook de key/columnname heeft een beperking, deze moet kleiner dan 64KB zijn. De laatste beperking die er toe doet: Alle data in één rij moeten kleiner zijn dan de schijfruimte op een enkele machine in een cluster, dat is alle data gekoppeld aan één rowkey.

4.2.2 Functionaliteit

Voor toegang van Cassandra zijn er voor verschillende talen high-level clients geschreven²³. Voor Java zijn dat onder andere Astyanax, Hector en Kundera. Daarnaast is er ook een mogelijkheid tot het gebruik van Hadoop. HBase heeft een Java API en ondersteuning voor Avro gateway APIs²⁴. Tevens is er een REST API aanwezig. Beide hebben ondersteuning voor Thrift.



Figuur 1 datamodel HBase, voor Cassandra geldt dat er geen meerder versies worden bijgehouden. Een enkel tableau geldt als datamodel voor Cassandra.

²¹ <http://wiki.apache.org/cassandra/DataModel>, http://en.wikipedia.org/wiki/Apache_Cassandra, <http://maxgrinev.com/2010/07/09/a-quick-introduction-to-the-cassandra-data-model/>

²² <http://hbase.apache.org/book.html#schema.creation>, <http://horicky.blogspot.com/2010/10/bigtable-model-with-cassandra-and-hbase.html>

²³ <http://wiki.apache.org/cassandra/ClientOptions>

²⁴ <http://en.wikipedia.org/wiki/HBase>

4.2.3 Betrouwbaarheid

Voor wide column stores bestaat een ring met nodes(servers). De nodes in de ring zijn op bepaalde manier gecodeerd²⁵, zo zijn er bepaalde codes voor datacenter, server en rack. Op deze manier is het mogelijk om voor een replication een ander rack, server of datacenter te kiezen.

Replication van data wordt bij wide column stores verzorgd door de onderliggende implementatie van HDFS²⁶. Hierbij wordt afhankelijk van de ingestelde replication factor gezorgd voor (in ieder geval) een kopie binnen hetzelfde rack, datacenter of continent. RackAwareStrategy is de plek belangrijk, met RackUnAwareStrategy maakt het niet uit welke node er beschreven wordt. Naast deze standaardondersteuning heeft HBase ook een eigen master-slave replication mogelijkheid welke asynchroon kan zorgen voor volledige replicatie van servers²⁷.

4.2.4 Robuustheid

Voor de robuustheid gebruiken de databases een commitlog. Dit is een file waarin alle writes van een bepaalde periode wordt geschreven, per periode wordt deze commit log in de database geschreven. Deze commitlog wordt met behulp van fsync²⁸ naar de schijf geschreven op een zo goed mogelijke manier. Indien het systeem crasht, kun je ten hoogste de data in de commitlog verliezen. Deze file kun je ook op een andere locatie opslaan, dat gaat echter wel ten koste van de snelheid. De periode van het flushen van de commitlog is flexibel aan te passen. Het is mogelijk om elke edit gelijk af te handelen, dit gaat wel ten koste van de performance. De data die in de database staat wordt niet door een crash of poweroff gewijzigd.

4.2.5 Ondersteuning

Cassandra is ontwikkeld door Facebook en wordt op dit moment gebruikt door Twitter, geen kleine spelers in de webwereld. Daarnaast zijn er verschillende bedrijven die ondersteuning verlenen voor Cassandra²⁹: Acunu, Datastax, Impetus, Onzra en Cubet. Dat is zonder twijfel voldoende ondersteuning.

Voor HBase³⁰ geldt hetzelfde, momenteel bieden enkele bedrijven op grote schaal commerciële ondersteuning voor HBase (en Hadoop) waaronder Cloudera en Sematext. Meerdere grote bedrijven passen HBase toe in productieomgevingen, waaronder Trend Micro, Mozilla, Twitter en Yahoo. Hierbij ligt de nadruk in veel gevallen op het verzamelen van zeer grote hoeveelheden data en het uitvoeren van analyse hierop.

4.2.6 Schaalbaarheid

HBase³¹ is volledig gebouwd op het Hadoop FileSystem (HDFS) waardoor horizontale schaalbaarheid vrij diep geïntegreerd in het systeem zit. Bijkomend nadeel hierbij is wel dat de databases een minimum van

²⁵ <http://wiki.apache.org/cassandra/Operations>

²⁶ http://hadoop.apache.org/common/docs/current/hdfs_design.html#Data+Replication

²⁷ <http://www.slideshare.net/benjaminblack/introduction-to-cassandra-replication-and-consistency>

²⁸ <http://pubs.opengroup.org/onlinepubs/7908799/xsh/fsync.html>

²⁹ <http://wiki.apache.org/cassandra/ThirdPartySupport>

³⁰ <http://www.slideshare.net/amansk/hbase-hadoop-day-seattle-4987041>

³¹ <http://goo.gl/BnBYS>, <http://wiki.apache.org/hadoop/Hbase>

vijf servers nodig, namelijk een NameServer heeft om de infrastructuur op te zetten en het geheel in de lucht te houden, hiermee valt het al snel af als oplossing voor kleinere projecten en bedrijven.

Cassandra kan wel op twee servers draaien³², echter is het dan de vraag of Availability of Consistency uit het CAP principe het belangrijkste is. Aangezien beide ten koste van de performance van de database gaat. Het aanbevolen aantal servers voor Cassandra is drie, dat heeft te maken met het schrijven van data en de replication factor. Als er een server uitvalt, kan het systeem nog door blijven werken, als er één server overblijft wordt dat heel lastig. Cassandra heeft ook een goede Hadoop integratie³³, wat zich uitstekend leent voor uitbreiding.

³² <http://wiki.apache.org/cassandra/MultinodeCluster>

³³ <http://wiki.apache.org/cassandra/HadoopSupport>

5. Afweging

Nu in hoofdstuk 4 een overzicht is gegeven van de fundamenteën van de verschillende databasesystemen die door de voorselectie zijn gekomen, zullen we vervolgens een afweging proberen te maken tussen de verschillende systemen om tot slot tot een meest geschikte kandidaat te komen.

5.1 Functionaliteit

MongoDB	CouchDB	HBase	Cassandra
++	+	+	+

Op het gebied van functionaliteit voldoen de in hoofdstuk 4 bekeken systemen stuk voor stuk. De eisen aan de functionaliteit, zoals het gebruiken van een database systeem in combinatie met een Java applicatie, zijn namelijk meegenomen in de voorselectie in hoofdstuk 3. Wel zijn er meer genuanceerde verschillen te vinden tussen de verschillende systemen. Zo bevat MongoDB een uitgebreide query engine waarmee op dynamische wijze queries uitgevoerd kunnen worden met indien gewenst vooraf aangemaakte indices op de collectie.

CouchDB aan de andere kant werkt met 'views' welke bestaan uit vooraf gedefinieerde MapReduce-opdrachten waarbij het resultaat wordt opgeslagen, dat komt de performance ten goede. Bij de overige systemen wordt er niet of minder gekeken naar de daadwerkelijk opgeslagen data waardoor er gekozen moet worden voor het gebruik van MapReduce wanneer de data inhoudelijk bekeken moet worden.

5.2 Betrouwbaarheid

MongoDB	CouchDB	HBase	Cassandra
++	+	++	++

Ook op het gebied van betrouwbaarheid zijn verschillen te vinden, hierbij hebben we in eerste instantie vooral gelet op melding van dataloss. Bij MongoDB zijn hier in het verleden veel meldingen van geweest, echter heeft MongoDB sinds versie 1.8 (uitgebracht in mei '11) journalling ingebouwd.³⁴ Met journalling zou bij toepassingen zonder replication en server crashes geen data verloren mogen gaan.

De meest recente melding van dataloss bij MongoDB lijkt een valse melding³⁵ van eind '11 te zijn. De meest actuele melding van dataloss bij CouchDB lijkt te zijn uit augustus '10 waarbij in de release van versie 1.0.0 een bug verstopt zat, deze data is uiteindelijk met behulp van een recovery tool terug te halen geweest³⁶. Voor Cassandra en HBase zijn geen recente meldingen van dataloss te vinden bij correct gebruik, wel lijkt het noodzakelijk voldoende aandacht te besteden aan de juiste configuratie van het cluster en het nauwkeurig volgen van de upgrade-instructies bij het vernieuwen van de versies.

³⁴ <http://www.mongodb.org/display/DOCS/Durability+and+Repair>

³⁵ <http://news.ycombinator.com/item?id=3202081>

³⁶ <http://couchdb.apache.org/notice/1.0.1.html>

5.3 Onderhoudbaarheid

	MongoDB	CouchDB	HBase	Cassandra
Licentie:	++	++	++	++
Aanpasbaarheid:	+	-	+	+
Server monitoring:	++	-	-	++
Stabiliteit:	+	+/-	++	+/-
Oordeel:	++/+	+/-	+	++/+

5.3.1 Licentie

De vier projecten zijn allemaal open source, in alle vier gevallen is de code is uitgegeven onder een 'copy left'-licentie, hetzij de AGPL (MongoDB)³⁷, hetzij de Apache License (overigen)^{38 39 40}. Hierbij verschilt de AGPL ten opzichte van de Apache License in die zin dat wanneer er een aangepaste AGPL-toepassing via een netwerk wordt aangeboden, de broncode hiervan beschikbaar moet zijn voor de gebruikers van dit systeem⁴¹. Hier is echter geen sprake van wanneer deze software wordt gebruikt maar niet direct wordt aangeboden. Dit is het geval met een SaaS-dienst die als opslag een losse DBMS gebruikt. Op dit punt scoren de verschillende systemen voor het beoogde gebruik dan ook gelijk.

5.3.2 Aanpasbaarheid

Naast een beschikbare broncode is ook de aanwezigheid van unit- of andersoortige tests een sterke pre. Wanneer er door de ontwikkelaars van het bedrijf aanpassingen aan het database systeem gemaakt worden, is het wenselijk om door middel van deze tests te kunnen controleren of andere delen van het systeem niet kapot zijn gegaan.

Elk van de vier projecten hebben een verzameling met unit-, integration- en/of behavioral-tests in hun code repository staan. Er zijn geen statistieken beschikbaar van de projecten omtrent code coverage, wel zijn er in alle repositories recente wijzigingen aan de code en corresponderende test cases te vinden. Het is hiermee aannemelijk dat de tests enige voorspellende waarde bevatten. Op het gebied van aanpasbaarheid hebben HBase en Cassandra tevens een streepje voor omdat deze zijn geschreven in Java, wat tevens de in-house programmeertaal is. MongoDB is geschreven in C++ en CouchDB in de wat minder bekende programmeertaal Erlang wat een drempel zou kunnen vormen voor het maken van aanpassingen.

5.3.3 Server monitoring

Een ander belangrijk kwaliteitsattribuut is de beheerbaarheid van de servers zelf. Binnen E.Novation wordt er gebruik gemaakt van Nagios om de status van systemen bij te houden. Dit is ook een van de criteria waarnaar gekeken is. Hierbij vallen MongoDB⁴² en Cassandra⁴³ positief op doordat er Nagios-

³⁷ <http://www.mongodb.org/display/DOCS/Licensing>

³⁸ <http://www.apache.org/licenses/>

³⁹ <http://hbase.apache.org/docs/r0.89.20100726/license.html>

⁴⁰ <http://cassandra.apache.org/>

⁴¹ <http://www.gnu.org/licenses/gpl-faq.html#AGPLv3InteractingRemotely>

⁴² <https://github.com/mzupan/nagios-plugin-mongodb>

plugins beschikbaar zijn met diepgaande statistieken (zaken als responsetijden, status van individuele servers, grootte van databases). Voor CouchDB en HBase lijken geen actieve en stabiele projecten te bestaan om statistieken verder dan uptime te verzorgen.

5.3.4 Stabiliteit

Onder onderhoudbaarheid verstaan wij ook de mogelijkheid tot het upgraden van het systeem naar nieuwere versies. Om hier een kwaliteitsoordeel over te kunnen geven hebben we gekeken naar de breaking changes sinds begin 2011.

Bij MongoDB wordt er voor het laatst over breaking changes gesproken bij upgrades naar de voorgaande versie (1.8), dit betreft een release uit mei 2011⁴⁴. In deze release zijn minimale wijzigingen doorgevoerd aan het teruggeven van de resultaten van MapReduce-queries.

CouchDB heeft een uitgebreid document met een lijst aan breaking changes⁴⁵ voor releases. Dit geldt ook voor de, op het moment van schrijven, aanstaande release van versie 1.2.0. Hierbij zijn enkele punten, zoals vernieuwde validatie op database-inserts die vroeger niet bestond, significant en kunnen voor problemen zorgen bij migraties wanneer de gebruikte applicaties hier niet goed op inspelen. Het gros van de wijzigingen betreft echter zaken die zich op administratief gebied afspelen en relatief weinig impact zullen hebben.

HBase heeft volgens hun wiki sinds versie 0.90, uitgekomen januari 2011⁴⁶, geen grote breaking changes doorgemaakt⁴⁷. Wel zijn er op deze wiki instructies te lezen voor upgrades vanaf de 0.20-versiereeks, hierbij zijn enkele configuratiebestanden aangepast welke gecorrigeerd moeten worden voordat er geüpgrade wordt om zo dataverlies te voorkomen.

Cassandra heeft sinds januari 2011 drie verschillende major releases gekregen⁴⁸, namelijk van 0.7 naar 0.8 en vervolgens naar versie 1.0. De release van versie 0.8 heeft meerdere wijzigingen met zich meegebracht die implicaties hebben voor applicaties en de databasecluster, onder andere configuratiebestanden zijn aangepast, klassen zijn verplaatst naar andere packages en loadbalancing is verplaatst naar een andere applicatie. De wijzigingen van 0.8 richting 1.0 zijn echter minimaal met slechts enkele configuratieopties die zijn hernoemd.

⁴³ <https://github.com/dmccnelis/NagiosCassandraMonitor>

⁴⁴ <http://www.mongodb.org/display/DOCS/1.8+Release+Notes>

⁴⁵ http://wiki.apache.org/couchdb/Breaking_changes

⁴⁶ <https://issues.apache.org/jira/browse/HBASE/fixforversion/12313607>

⁴⁷ <http://wiki.apache.org/hadoop/Hbase/HowToMigrate>

⁴⁸ <https://issues.apache.org/jira/browse/CASSANDRA#selectedTab=com.atlassian.jira.plugin.system.project:changelog-panel&allVersions=true>

5.4 Bruikbaarheid

	MongoDB	CouchDB	HBase	Cassandra
Concept:	++	++	+/-	+/-
Documentatie:	++	+	+	+
Oordeel:	++	++/+	+	+

5.4.1 Duidelijkheid concept

Het principe van een document store is eenvoudig voor te stellen door het te zien als een key-value map. Een principe waar veel developers bekend mee zullen zijn. Deze ligt ook dicht bij de manier van werken van een traditionele relationele database. Het datamodel van een wide column store daarentegen is enigszins ingewikkeld. Waar document stores een relatief uniforme structuur hebben (een map met keys/values en mogelijk maps als values), hebben de wide column stores (WCS) een minder flexibele gelaagdheid. De WCS werken met meerdere hiërarchieën en dat is minder triviaal dan de bekende structuur van de document stores.

Dit betekent echter niet dat WCS-implementaties per definitie een slechte keus zijn, wanneer de gebruiker bekend is met de wijze van opslag en de bijbehorende voor- en nadelen zal de toepassing geen groot probleem meer mogen vormen. Wel zullen document stores minder uitleg nodig hebben dan de WCS.

5.4.2 Documentatie

De faciliteiten om de genoemde systemen onder de knie te krijgen zijn over het algemeen sterk te noemen. Zowel MongoDB als CouchDB hebben goede documentatie.^{49 50} De documentatie wordt regelmatig bijgewerkt met informatie over de laatste systeemversies. Ook bij de WCS hebben de ontwikkelaars hun best gedaan goede tutorials en documentatie^{51 52} te verzorgen. Op dit vlak zijn enkele verschillen te vinden. Er zijn vooral verschillen te vinden in de helderheid van de documentatiewiki van MongoDB en de wikis van andere projecten. De wiki van MongoDB bevat een heldere navigatiestructuur, bij de wikis van andere projecten is dit minder het geval. Hierdoor is informatie op de wiki van MongoDB te vinden zonder dat de gebruiker specifiek de zoekfunctie moet gebruiken, dit maakt het eenvoudiger om gerelateerde documenten door te lezen. Daarnaast geeft het een helder overzicht van wat er beschikbaar is.

⁴⁹ <http://www.mongodb.org/display/DOCS/Home>

⁵⁰ <http://couchdb.apache.org/docs/>

⁵¹ <http://wiki.apache.org/hadoop/Hbase>

⁵² <http://cassandra.apache.org/>

5.5 Vervangbaarheid

MongoDB	CouchDB	HBase	Cassandra
-	-	-	-

De vervangbaarheid van de database wil zeggen dat de mogelijkheid bestaat om het ene databasesysteem voor een andere te vervangen. Dit bij voorkeur zonder implicaties voor de bestaande applicatie, zodat er een mogelijkheid is om zonder de applicatie aan te passen over te stappen naar een alternatief database systeem.

Doordat NoSQL databases over het algemeen geen ondersteuning hebben voor een universele querytaal is er niet vanzelfsprekend de mogelijkheid om zonder gevolgen het systeem te vervangen door een alternatief. Bij de NoSQL databases is er in enkele gevallen wel ondersteuning voor toegang tot de database door middel van een REST API⁵³.

Doordat er een uniforme toegangsmogelijkheid bestaat, namelijk door middel van deze interface, is het in theorie mogelijk om de applicatie met een andere API te laten communiceren zonder verdere wijzigingen. De praktijk blijkt echter minder ideaal, zowel CouchDB⁵⁴ als HBase⁵⁵ bieden standaard een REST API, het blijkt echter dat deze APIs volledig incompatible zijn door de afwijkende onderliggende opslag en bijbehorende manier van het opstellen van queries.

5.6 Efficiëntie

	MongoDB	CouchDB	HBase	Cassandra
Middelenbeslag:	+	+	+	+
Schaalbaarheid:	++	+	++	++
Oordeel:	++/+	+	++/+	++/+

Om een oordeel te geven over de relatieve efficiëntie in het gebruik van resources tussen de verschillende systemen hebben we conform de kwaliteitsattributen gekeken naar het middelenbeslag en de schaalbaarheid van de systemen.

5.6.1 Middelenbeslag

Omdat het lastig is op basis van een literatuurstudie te beoordelen in hoeverre de belasting van de systemen onderling verschillen op het gebied van middelenbeslag (resource usage) hebben we hierbij gekeken naar meldingen van (zeer) hoog geheugen- en/of processorgebruik op de issue trackers⁵⁶ van

⁵³ http://en.wikipedia.org/wiki/Representational_State_Transfer

⁵⁴ http://wiki.apache.org/couchdb/HTTP_Document_API

⁵⁵ <http://wiki.apache.org/hadoop/Hbase/Stargate>

⁵⁶ <https://issues.apache.org/jira/secure/IssueNavigator.jspa?>
<https://jira.mongodb.org/secure/IssueNavigator.jspa>

de betreffende applicatie. Daarnaast hebben we gekeken naar specifieke designkeuzes uit de verschillende systemen die voor lagere efficiëntie zouden kunnen zorgen zoals de global write lock in MongoDB en het verplichte onderhoud bij Cassandra.

Een type fout met mogelijke gevolgen voor de systeemstabiliteit die voorkomt bij hoofdzakelijk langdraaiende applicaties zijn “memory leaks”. Als een applicatie ‘memory leaked’ wil dat zeggen dat de applicatie geheugen in gebruik neemt maar dit niet meer vrijgeeft waardoor het totale geheugengebruik op zal blijven lopen met mogelijke performanceproblemen als gevolg. Om enig inzicht te krijgen in de veelvoorkomendheid van dit soort geheugengerelateerde problemen hebben wij de issuetrackers van de vier databases doorzocht over het aantal issues met betrekking tot memory leakage.

Hierbij is gebleken dat Cassandra en CouchDB in de periode januari tot en met maart 2012 relatief weinig issues hebben gekend welke raakvlak hebben met deze problematiek, MongoDB kende hier al een hoger aantal en HBase heeft hiervan de grootste hoeveelheid issues staan.

Op een zelfde manier is gekeken naar de CPU freezes en exorbitant hoog processorgebruik in de databases. Een freeze zorgt ervoor dat een programma niet meer reageert, aan de andere kant is ook een constant hoog processorgebruik een mogelijke indicatie van een fout in de programmatuur. Voor deze problemen geldt ongeveer dezelfde verdeling wat betreft de issues, Cassandra en CouchDB hebben erg weinig issues wat dit onderwerp betreft terwijl HBase en MongoDB er meer lijken te hebben.

Hierbij moet opgemerkt worden dat de hoeveelheden open issues van beide typen zeer sterk fluctueren door de tijd. Er kunnen dan ook niet zonder meer statistisch verantwoorde conclusies getrokken worden op basis van deze data zonder rekening te houden met de totale hoeveelheid issues, de bereidheid van de gebruikers om tickets aan te maken voor problemen van deze aard en natuurlijk de snelheid waarmee dergelijke zaken worden opgelost. Deze zaken vallen echter buiten de scope van ons onderzoek en zijn dan ook niet meegenomen, dit onderdeel weegt dan ook slechts deels mee in de uiteindelijke besluitvorming.

Naast mogelijke bugs zijn er ook nog een aantal ontwerpkeuzes te benoemen die voor verlaagde efficiëntie kunnen zorgen. Zo heeft MongoDB een global write lock⁵⁷, dat is een lock die in werking treedt zodra er een ‘page fault’ optreedt bij het lezen of schrijven van gegevens. Hierbij moet de permanente opslag geraadpleegd worden. In oudere versies van MongoDB werden tijdens het raadplegen van de opslag geen andere processen afgehandeld. Tijdens het raadplegen van de harde schijf heeft de server niets te doen en duurt de transactie afhankelijk van het gebruikte opslagmedium enige tijd langer.

Om de hoeveelheid page faults te voorkomen kan er gebruik gemaakt worden van sharding. Op die manier kan er op elke server een deelverzameling van de data beheerd worden en op die manier in totaal een groter deel van de dataset in geheugen geladen kan worden waardoor er minder vaak van disk gelezen zal worden.

⁵⁷ <http://www.mongodb.org/display/DOCS/How+does+concurrency+work>

In versie 2.0 van MongoDB is de werking van de write lock veel verbeterd⁵⁸, in de nieuwe versie is ‘yielding’ toegevoegd tijdens de lock om op deze manier tijdens deze wachttijd andere acties uit te kunnen voeren waardoor de prestaties pas bij zeer hoge hoeveelheden writes omlaag gaan. Bij de overige systemen zijn geen vergelijkbare limitatie te vinden.

Cassandra heeft een verplichte onderhoudstaak die periodiek uitgevoerd moet worden en een sterke invloed kan hebben op de performance van de betreffende server⁵⁹. Wanneer dit onderhoud niet uitgevoerd wordt bestaat de kans dat verwijderacties worden ‘vergeten’ door het cluster en data ongewenst weer terug in de database terecht komt.

5.6.2 Schaalbaarheid

Op het gebied van schaalbaarheid verschillen de vier databases tot op zekere hoogte van elkaar. Hierbij vloeien de verschillen vooral voort uit de ontwerpkeuzes van de datamodellen. Zo zijn de wide column stores bij uitstek gemaakt om gedistribueerd over vele servers te werken en hierbij hoge write performance te garanderen. Dit heeft als consequentie dat leesacties in sommige gevallen minder snel zullen zijn dan schrijfacties, afhankelijk van de mate van spreiding van de data en de hoeveelheid servers die hierbij geraadpleegd moeten worden.

Document stores als MongoDB werken niet met een dergelijke decentrale aanpak maar maken bijvoorbeeld gebruik van een primary server om de writes te registreren en doorspelen naar de overige servers. Om toch de schrijfperformance te kunnen schalen werken deze systemen met sharding waarbij documenten op basis van een index verspreid in stukken verdeeld worden over de beschikbare servers waarmee schrijf- en zoekacties zich specifiek op een deel van de servers kunnen richten. CouchDB heeft standaard geen ondersteuning voor sharding, wel zijn er meerdere third-party tools beschikbaar om dit mogelijk te maken.

⁵⁸ <http://blog.pythonsito.com/2011/12/mongodb-write-lock.html>

⁵⁹ <http://wiki.apache.org/cassandra/Operations#Consistency>

6. Conclusie

Nu de verschillende systemen getoetst zijn op de opgestelde kwaliteitsattributen kan het totaal opgemaakt worden. Dit doen we aan de hand van de beoordelingen die in hoofdstuk 5 zijn gegeven. Hierbij wordt in de optelling onderscheid gemaakt tussen de primaire en secundaire kwaliteitsattributen, de primaire eisen wegen zwaarder voor het bedrijf en deze punten hebben dan ook een dubbele weging gekregen in de optelling. Wanneer we alle beoordelingen op rij zetten, krijgen we het volgende overzicht:

	MongoDB	CouchDB	HBase	Cassandra
Primaire kwaliteitseisen				
Functionaliteit:	++	+	+	+
Betrouwbaarheid:	++	+	++	++
Onderhoudbaarheid:	++/+	+/-	+	++/+
Secundaire kwaliteitseisen				
Bruikbaarheid:	++	++/+	+	+
Vervangbaarheid:	-	-	-	-
Efficiëntie:	+	+	+	+
Eindoordeel:	++/+	+	+	+

In de tabel is zichtbaar dat de databases in het algemeen dicht bij elkaar liggen qua beoordeling. Dat is ook deels logisch door de voorselectie aan het begin van het proces waardoor de opties die lager zouden scoren niet meegenomen zijn in de beoordeling. Met de eindbeoordeling steekt MongoDB net iets boven de andere drie kandidaten uit, dit systeem scoort op alle fronten goed. Cassandra en HBase scoren vergelijkbaar goed en worden gevolgd door CouchDB waarbij de uiteindelijke scores van de vier systemen in de kleine details zitten.

Onze conclusie luidt dat MongoDB het beste scoort bij de opgestelde lijst van kwaliteitsattributen en dan ook op basis van deze informatie de meest interessante optie lijkt om toe te passen binnen het bedrijf. MongoDB scoort op alle punten gelijk of beter dan de concurrentie in de test, hoewel de afstand tot Cassandra slechts beperkt is. Gezien de te verwachten usecase van E.Novation, zal er waarschijnlijk niet binnen afzienbare tijd gewerkt gaan worden met een dataset die groot genoeg is om daadwerkelijk te profiteren van de voordelen die Cassandra in dergelijke situaties biedt. MongoDB lijkt ook hier dan als beste keuze uit te komen.

Bijlage A: Onderzochte systemen

De suggesties voor de verschillende bestaande databases die wij in dit onderzoek hebben gebruikt komen van nosql-database.org⁶⁰ en de wikipedia pagina⁶¹ van NoSQL. In de tabel hieronder staan de databases gesorteerd op verschillende soorten databases die in eerste instantie relevant waren. Na een vooronderzoek zijn er nog enkele overgebleven, zoals u kunt lezen in dit document.

Wij hebben de volgende systemen meegenomen in onze selectie:

Key-value	Document store	Wide column stores
Amazon DynamoDB MEMBASE Redis Voldemort MemcachedB	Citrusleaf Clusterpoint Server CouchDB Jackrabbit MongoDB Persevere RavenDB Riak SisoDB Terrastore ThruDB	Amazon SimpleDB Cassandra HBase Hypertable

<http://aws.amazon.com/dynamodb/>; <http://www.couchbase.com/wiki/display/couchbase/Home>;
<http://redis.io/>; <http://project-voldemort.com/>; <http://memcachedb.org/>

<http://citrusleaf.net/>; <http://www.clusterpoint.com/>; <http://couchdb.apache.org/>;
<http://jackrabbit.apache.org/>; <http://www.mongodb.org/>; <http://persvr.org/>;
<https://github.com/ravendb/ravendb>; <http://wiki.basho.com/>; <http://www.sisodb.com/>;
<http://code.google.com/p/terrastore/>; <http://code.google.com/p/thruDB/>

<http://aws.amazon.com/simpledb/>; <http://cassandra.apache.org/>; <http://hbase.apache.org/>;
<http://hypertable.org/>

⁶⁰ <http://nosql-database.org/>

⁶¹ <http://en.wikipedia.org/wiki/NoSQL>

Bijlage B: Woordenlijst

AGPL - Affero General Public License⁶², open source licentie waarmee de software aangepast en verkocht mag worden, mits het dit recht wordt doorgegeven en de auteurs vermeld worden. Met extra paragraaf over verspreiding via het netwerk.

Apache License⁶³ - Software mag aangepast en verspreid worden, mits een kopie van de Apache License wordt toegevoegd en de eerder toegevoegde copyright vermeldingen blijven bestaan.

API - Application Programming Interface, manier om functionaliteit van een applicatie te gebruiken zonder de implementatie details van de applicatie te kennen.

BigTable - proprietair DBMS van Google

DBMS - database managementsysteem

Document store - DBMS waarbij de data in de vorm van een document met attributen en waarden wordt opgeslagen.

MapReduce - Slimme manier van het opdelen van een probleem in kleinere subproblemen, waar alle antwoorden vervolgens verzameld worden en een samengesteld antwoord als output heeft.

RDBMS - relationeel DBMS waarin data wordt opgeslagen in de vorm van tabellen met een vast aantal kolommen en rijen voor de op te slaan data

Replication - delen van informatie op een dusdanige manier dat de duplicaten van de informatie, op meerdere locaties, consistent blijven. Het doel is het bevorderen van betrouwbaarheid en beschikbaarheid.

Repository - een centrale locatie voor opslag van een softwareproject dat met meerdere andere clients gedeeld is.

SaaS - Software as a Service, een manier van IT-dienstverlening waarbij software als dienst wordt aangeboden, vaak tegen een maandelijks tarief, waarbij de klant niet verantwoordelijk is voor het onderhoud en de aanschaf van de software.

Sharding - onderverdeling van data over verschillende servers, met als doel betere prestaties.

Wide column store - DBMS volgens de theorie uiteengezet in een research paper over Google's BigTable systeem⁶⁴.

⁶² <http://www.gnu.org/licenses/agpl-3.0.html>

⁶³ <http://www.apache.org/licenses/LICENSE-2.0>

⁶⁴ <http://research.google.com/archive/bigtable.html>

Bijlage E: Implementatieanalyse

Bachelorproject 2012

Voor de Bachelor Technische Informatica, TU Delft

Maarten van Meijeren, Yorick Holkamp

In samenwerking met *E.Novation*

Begeleiders

Bastiaan Bakker (E.Novation, afdeling Application Development)

Jan Hidders (TU Delft, afdeling Web Information Systems)



Samenvatting

Dit document bevat een onderzoek dat het verschil in ontwikkeltijd en code complexiteit van twee applicaties beschrijft. De eerste applicatie past de patterns Command Query Responsibility Segregation (CQRS) en Event Sourcing toe en maakt gebruik van MongoDB als opslagsysteem. De tweede applicatie is opgebouwd volgens de meer traditionele Layered Architecture-aanpak (LA) en gebruikt een relationele database. Beide applicaties implementeren dezelfde set functionaliteit en maken gebruik van het Spring MVC framework om een adresboek in de vorm van een webapplicatie te bieden.

Als onderdeel van het onderzoek zijn twee sets features toegevoegd, namelijk een zoekfunctie op basis van naam voor de contacten in het adresboek en de mogelijkheid om in plaats van slechts één telefoonnummer een n-aantal telefoonnummers op te kunnen slaan. Tijdens de ontwikkeling van deze functies is de benodigde ontwikkeltijd bijgehouden en is na afloop de code complexiteit van beide systemen vergeleken. Tot slot is er een vergelijking gemaakt met een onderzoek wat enkele raakvlakken heeft met dit project.

Uit de resultaten blijkt dat in ons geval de leercurve van een CQRS/ES-systeem steiler verloopt dan die van de meer standaard layered architecture. Ook wijzen de resultaten en ervaringen op de mogelijkheid dat het implementeren van nieuwe features in een CQRS/ES-systeem, na een korte leercurve, sneller zou kunnen verlopen doordat er met de meer opgesplitste architectuur minder kennis over het gedrag van het gehele systeem vereist is. Tot slot blijkt ook dat er met het gebruik van CQRS/ES een grotere hoeveelheid methoden en klassen benodigd is om het pattern invulling te geven.

Inhoud

Samenvatting	2
Inleiding.....	4
Analyse tijdmeting	5
Case 1: Toevoegen van eenvoudige (zoek)functionaliteit	5
Case 2: Introduceren van een wijziging in het systeem.....	7
Conclusie	8
Analyse codekwaliteit	9
Vergelijking	9
Complexiteit binnen klassen	9
Complexiteit binnen methoden	10
Conclusie	10
Vergelijking van conclusie met Sogyo onderzoek.....	11
Ervaringen ontwikkeling	12
Maven dependency management	12
Gebruik van Spring.....	13
Object relational mappers	13
Conclusie	13
Aanbevelingen	14
Conclusie	15
Bijlage A: Tijdmetingen	16
Bijlage B: Woordenlijst.....	17

1. Inleiding

Als onderdeel van het bachelorproject zijn er twee applicaties opgebouwd. Een op basis van een layered architecture (LA) met een relationele database als opslag en een systeem gebaseerd op Command Query Responsibility Segregation (CQRS) met Event Sourcing en MongoDB¹ als opslagsysteem. Nadat deze twee projecten in vergelijkbare staat gebracht zijn, zijn er features toegevoegd. Zo is er een zoekfunctie voor contacten aan beide systemen toegevoegd. Daarnaast zijn de contactpersonen aangepast zodat deze niet één maar meerdere telefoonnummers kunnen bevatten.

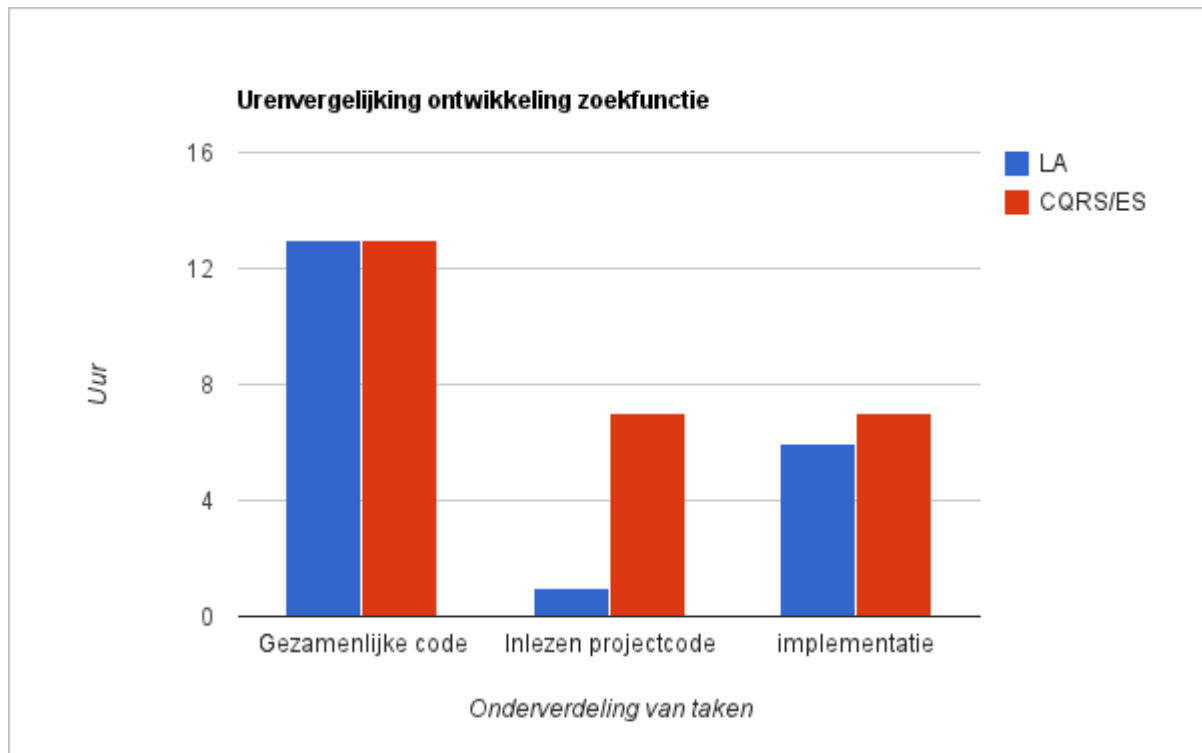
Tijdens deze aanpassingen is de hiervoor benodigde tijd gemeten. De functionaliteit is in beide applicaties geïmplementeerd. Dit is door dezelfde developer gedaan, omdat er anders geen eerlijke vergelijking gemaakt kan worden. Ook is er een vergelijking gemaakt tussen de complexiteitscijfers van beide projecten. Wanneer er op later moment tijd beschikbaar komt en dit voor het bedrijf voldoende prioriteit heeft, kunnen hier nog meer wijzigingen aan toegevoegd worden. Met meer data is het uiteindelijk mogelijk om tot een statistisch significant resultaat te kunnen komen. Tot die tijd geven de resultaten vooral een indicatie van de benodigde tijd en levert het vooral ervaring op met de gebruikte technieken.

¹ <http://www.mongodb.org/>

2. Analyse tijdmeting

2.1. Case 1: Toevoegen van eenvoudige (zoek)functionaliteit

Een feature die we met de start van het project hebben bedacht is het toevoegen van een zoekfunctie voor contactpersonen aan het systeem. De implementatie bestaat voor een deel uit code die overeenkomt tussen beide systemen. Zoals code voor de controllers en views van Spring MVC#. Maar ook voor een deel aan systeemspecifieke zaken, bijvoorbeeld contact met de database. De gemeenschappelijke delen zijn echter minder interessant in de vergelijking dan systeemspecifieke delen. Deze zijn dan ook zoveel mogelijk los van elkaar gemeten.



Figuur 1: Gespendeerde tijd voor het toevoegen van de zoekfunctie. De tabelvorm staat in bijlage A.

De meetresultaten in figuur 1 laten zien dat het toevoegen van features binnen de CQRS-variant significant meer tijd kost dan binnen de LA-variant. Deze tijd zit vooral in de leercurve die doorlopen moet worden om aan de slag te kunnen met CQRS-project. Te verwachten is dat deze investering slechts eenmalig is. Want zodra de ontwikkelaar bekend is met de werking van Axon# en het CQRS-pattern zal het geheel vloeiender verlopen. Toekomstige wijzigingen zullen hier minder hinder van ondervinden.

Los van de uren die in figuur 1 uiteen gezet zijn, is er tijdens de ontwikkeling aan het CQRS-project veel tijd verloren gegaan aan configuratieproblemen rondom het Spring-framework. Bij dit project werd er veelvuldig gebruik gemaakt van de 'autowiring'-feature welke automatisch allerlei properties van 'JavaBeans' kunnen instantiëren, maar daarmee ook een zeer fragiel geheel vormen.

Zo zijn er na het toevoegen van een aangepaste repository voor het view-model problemen opgetreden in de configuratie van andere repositories. Het oplossen van deze problemen heeft vrij veel tijd gekost welke niet direct relevant is voor de vergelijking. Echter geeft dit wel aan dat de Autowiring-feature niet de meest robuuste en eenvoudig te debuggen configuratie-oplossing biedt.

Een voorbeeld van een ontvangen foutmelding is het volgende:

```
org.springframework.beans.factory.BeanCreationException: Error creating bean with name
'nl.enovation.addressbook.cqrs.webui.controllers.ContactsControllerIntegrationTest':
Injection of autowired dependencies failed;

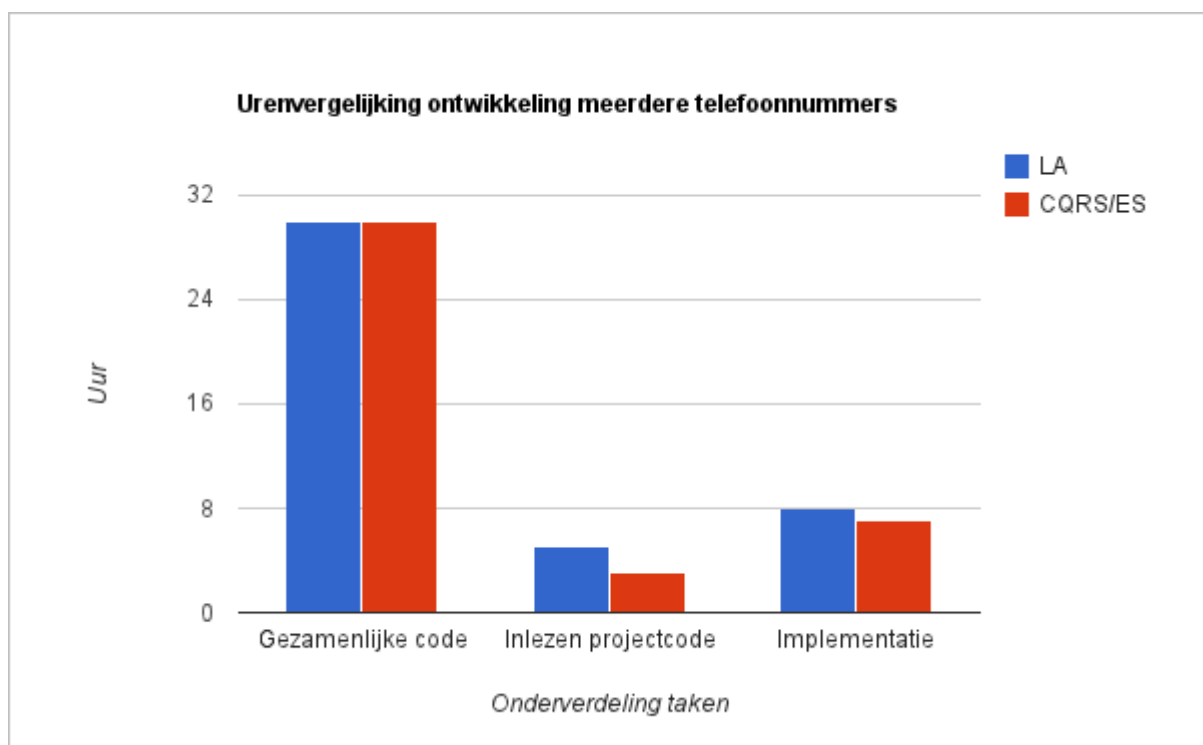
    nested exception is org.springframework.beans.factory.BeanCreationException: Could
    not autowire field: private
    nl.enovation.addressbook.cqrs.webui.controllers.ContactsController
    nl.enovation.addressbook.cqrs.webui.controllers.ContactsControllerIntegrationTest.c
    ontroller;
        nested exception is
        org.springframework.beans.factory.NoSuchBeanDefinitionException: No matching
        bean of type
        [nl.enovation.addressbook.cqrs.webui.controllers.ContactsController] found
        for dependency: expected at least 1 bean which qualifies as autowire
        candidate for this dependency. Dependency annotations:
        {@org.springframework.beans.factory.annotation.Autowired(required=true)}
```

Hierbij wordt er aangegeven dat er door een typefout van de ontwikkelaar in een van de XML-configuratiebestanden geen bean gevonden kan worden van het type ContactsController. Het probleem voor de minder ervaren ontwikkelaar zit hem hierbij in het gebrek aan inzicht in hoe het systeem precies tot dit punt komt. De voor ons gangbare stappen om fouten gerelateerd aan het Autowiring-systeem te debuggen bestaan vervolgens uit het handmatig nalopen van de configuratiebestanden in de hoop hier een typefout of verkeerde declaratie te ontdekken.

2.2. Case 2: Introduceren van een wijziging in het systeem

Als tweede case hebben we het wijzigen van een domeinmodel gekozen. Bij deze wijziging wordt een attribuut, in dit geval het telefoonnummer van de persoon, vervangen door een 1:n-relatie. Een contact kan nu dus een n-aantal telefoonnummers hebben. Het resultaat van deze aanpassing is tevens in een latere fase te gebruiken om het effect van veranderingen aan het domeinmodel en de bijbehorende datamigratie te bepalen.

Net als bij de eerdere besproken aanpassingen zijn de wijzigingen aan beide projecten door dezelfde developer doorgevoerd. Ook is de tijd die besteed is aan code die voor beide projecten nagenoeg gelijk is, zover mogelijk apart bijgehouden. Dit heeft vervolgens de resultaten van figuur 2 opgeleverd.



Figuur 2: Gespendeerde tijd voor het doorvoeren van de veranderingen aan het domeinmodel. In tabelvorm te vinden in Bijlage A.

Uit de resultaten blijkt dat het inlezen in de projectcode van het LA-systeem meer tijd heeft gekost dan bij het CQRS-project. Het verschil valt deels te verklaren door de mindere bekendheid van de ontwikkelaar met het LA-project. Tevens wijst het er mogelijk op dat het relatief eenvoudig is om nieuwe losstaande zaken door te voeren in het CQRS-systeem. Door losse koppelingen in het CQRS-pattern hoeft de developer namelijk niet heel het systeem te kennen, bij een LA-systeem is dat wel zo.

Ook de tijd benodigd voor de daadwerkelijke implementatie van de functionaliteit valt lager uit voor het CQRS-project. Bij deze aanpassingen is er zeker profijt te halen uit de losse koppeling tussen de verschillende delen van het systeem. Dit maakt zowel de ontwikkeling als het testen van de nieuwe functionaliteit eenvoudiger.

2.3. Conclusie

Tijdens het project zijn uiteindelijk een tweetal featuresets geïmplementeerd waarbij een flinke hoeveelheid ervaring opgedaan en gedocumenteerd is. De focus heeft minder op het produceren van een grote hoeveelheid featuresets gelegen. Daardoor zijn er minder datapunten gegenereerd en is de hoeveelheid data hiermee te beperkt gebleven om enige statistisch onderbouwde uitspraken te kunnen doen.

Wel is er veel ervaring opgedaan met beide systemen. Hierbij heeft het werk bij case 1 aangetoond dat de leercurve van CQRS in ons geval sterk steiler is verlopen dan de LA-tegenhanger waarmee er dus relatief veel tijd geïnvesteerd moest worden voordat er daadwerkelijk gebouwd kan worden. Uit case 2 is gebleken dat voor bepaalde wijzigingen er bij een CQRS-systeem minder systeemkennis nodig is om het geheel af te ronden. Uit beide cases is gebleken dat de benodigde implementatietijd nagenoeg gelijk is voor aanpassingen aan beide systemen.

De verschillen lijken op basis van onze metingen dan ook te zitten in de benodigde tijd om bekend te worden met het CQRS-principe en het Axon framework, wanneer er vervolgens ervaring is opgedaan zou deze tijd af kunnen nemen of gelijkwaardig blijven aan systemen met een layered architecture. Echter is dit een indicatie aan de hand van beperkte hoeveelheid data. Om te kunnen bepalen waar deze trend uiteindelijk precies op zal wijzen zal kunnen blijken uit een grootschaliger project. Met een groter aantal losstaande wijzigingen en een grotere groep proefpersonen zijn er meer datapunten te genereren, waarna een conclusie volgt die statistisch beter onderbouwd is.

3. Analyse codekwaliteit

Binnen E.Novation wordt de applicatie 'Sonar'² veelvuldig gebruikt om de kwaliteit van de code binnen verschillende projecten te monitoren. Sonar genereert een aantal interessante statistieken aan de hand van bekende tools als 'pmd'³, 'checkstyle'⁴ en een collectie van door de gebruiker aanpasbare metriekeken.

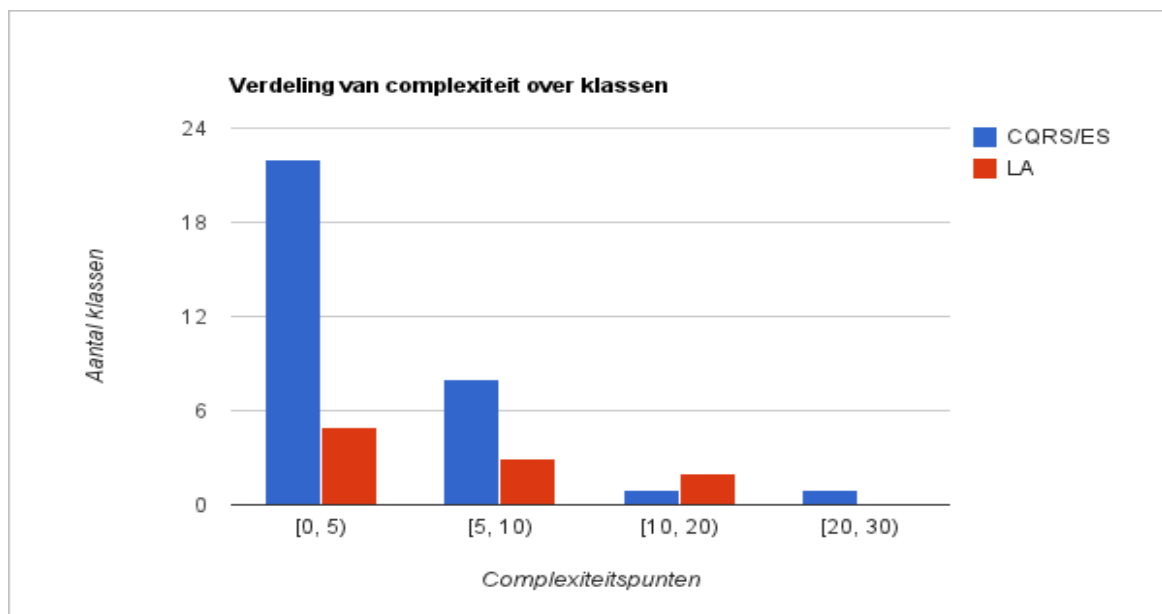
De metriek waar we vooral in geïnteresseerd zijn is de cyclomatic complexity⁵ van de broncode van beide applicaties. Dit geeft namelijk een indicatie van de complexiteit en daarmee de testbaarheid van de applicatie gegeven. De vraag is vervolgens of het gebruik van een alternatief design pattern voor minder complexe code zorgt.

3.1. Vergelijking

Om een vergelijking tussen beide projecten op basis van hun cyclomatic complexity te maken hebben we de gegevens gegenereerd door Sonar gebruikt, hier komen vervolgens twee grafieken uit voort.

3.1.1. Complexiteit binnen klassen

In de als eerste besproken metric van Sonar komt de verdeling van complexiteit over de klassen naar voren. Het eerste wat hierbij in het oog springt, is het grote verschil in de totale hoeveelheid klassen tussen beide projecten: Het CQRS/ES-project heeft in totaal 32 klassen tegenover 11 voor het LA-project. Deze grote hoeveelheid klassen valt te verklaren door de aanwezigheid van vele events en commands. Deze zijn door Axon vereist en verzorgen de data en control flow binnen de applicatie. Het zijn veel klassen omdat elk event en command een eigen klasse vormt in het systeem.



Figuur 3: Cyclomatic complexity van de klassen in beide projecten na implementatie.

² <http://www.sonarsource.org/>

³ <http://pmd.sourceforge.net/pmd-5.0.0/>

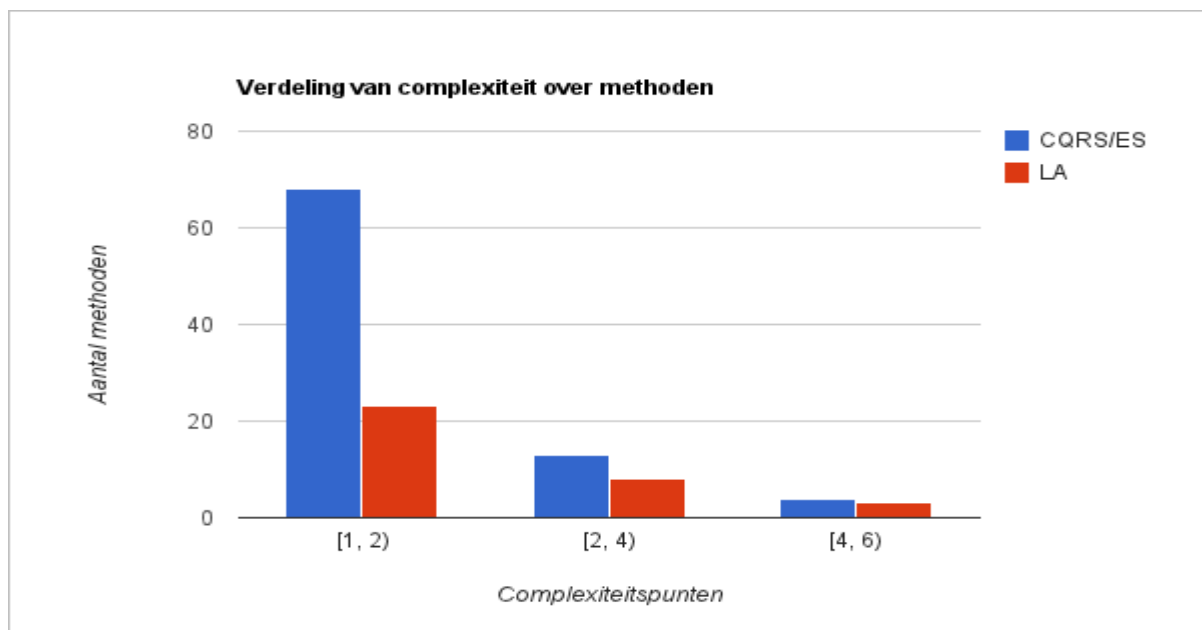
⁴ <http://checkstyle.sourceforge.net/>

⁵ McCabe (December 1976). "A Complexity Measure". IEEE Transactions on Software Engineering: 308–320, <http://www.literateprogramming.com/mccabe.pdf>

De aanwezigheid van al deze relatief eenvoudige klassen komt ook terug in de verdeling van complexiteit. Het is duidelijk zichtbaar dat de verdeling van de klassen binnen het CQRS project geconcentreerd ligt rond de klassen met zeer lage complexiteit (van 1 tot 5). Bij het LA-project is dit minder zichtbaar aanwezig en lijkt de verdeling van complexiteit meer lineair te verlopen.

3.1.2. Complexiteit binnen methoden

Naast de verdeling van complexiteit tussen de verschillende klassen geeft Sonar tevens een verdeling van functies over de verschillende complexiteitsniveaus. Deze verdeling is te zien in grafiek Figuur 4. Vergelijkbaar met grafiek Figuur 3 valt hier het verschil in aantallen op, het totaal ligt voor het CQRS/ES-project op 77 terwijl het LA-project slechts 35 methoden bevat.



Figuur 4: Cyclomatic complexity van methoden in beider projecten na implementatie

Dit verschil komt voornamelijk door de hoeveelheid eerder genoemde klassen die het Axon Framework nodig heeft om de flow binnen de applicatie te regelen. Zo heeft elk event en command zijn eigen klasse, daarnaast elk constructors (vaak een of twee per command en een per event), getters (vaak een per event of command) en evenzoveel event- en commandhandler-methoden om te anticiperen op deze objecten. Dit alles zorgt voor grote hoeveelheid eenvoudige methoden die de complexiteit van het project maar weinig beïnvloeden maar wel voor een relatief grote hoeveelheid extra code zorgen.

3.2. Conclusie

Uit de grafieken blijkt dat de complexiteit van zowel methoden als klassen in beide projecten relatief laag is. Beide projecten bevatten slechts enkele uitschieters met een complexiteit tussen de 10 tot 30 punten in de klassen. Alle twee de gevallen komen voort uit een controller van de webinterface. De complexiteit van methoden blijkt op gelijke wijze vrij laag te zijn, de methoden hebben een maximale cyclomatic complexity tot en met 5 punten. Dit laat zien dat er in deze projecten geen zeer complexe zaken in één enkele methode worden afgehandeld. Dit ligt ook in de lijn der verwachting gezien de beperkte complexiteit van de applicaties. Ook enige nadruk op het afleveren van kwalitatief goede software draagt daar aan bij.

Wel blijkt dat het gebruik van CQRS/ES een zekere hoeveelheid aan code met zich meebrengt die niet in het project aanwezig dient te zijn. Deze code doet op zichzelf eigenlijk niets tot weinig. De vraag is uiteindelijk of deze code het geheel niet juist complexer maakt dan dat het voor meer overzicht zorgt. In ons geval, waarbij we enkel naar een vrij eenvoudige applicatie kijken, blijft de complexiteit van de layered architecture-applicatie binnen de perken. Er zijn dus geen complexe methoden nodig om de business logic binnen de applicatie af te handelen.

Mocht er uiteindelijk wel naar een applicatie gekeken worden die wel een complexere business logic bevat, dan is het mogelijk dat de voordelen van de opsplitsing, zoals CQRS die kent, meer aan het licht komen. Doordat CQRS een opsplitsing van de verschillende ‘concerns’ binnen de applicatie (de command- en query-kant) voorschrijft brengt dit als voordeel met zich mee dat de complexiteit mogelijk minder geconcentreerd zal zijn en daarmee een eenvoudiger testbaar geheel op zou kunnen leveren.

4. Vergelijking van conclusie met Sogyo onderzoek

Deze manier van onderzoek doen tussen twee verschillende software patterns is niet uniek. Ons onderzoek heeft raakvlakken met een ander onderzoek dat is gedaan in opdracht van Sogyo en gepresenteerd op NLJUG J-Fall 2010⁶. Hierbij heeft een student van de UvA een onderzoek voor zijn masterscriptie gedaan waarbij twee projecten, een ‘standaard’ layered architecture- en een Event Driven Architecture-applicatie (EDA), aan vier proefpersonen werden voorgelegd waarbij vervolgens in beide projecten wijzigingen geïmplementeerd werden.

Hoewel er overeenkomsten te zien zijn tussen beide onderzoeken komen ze niet helemaal overeen met de methode gebruikt bij Sogyo. Zo is de ontwikkeling van features niet door externe proefpersonen uitgevoerd maar door onszelf. Ook zijn de wijzigingen slechts door één student per set doorgevoerd om de beperkte tijd die beschikbaar is optimaal te benutten. Daarnaast gebruiken wij het CQRS/ES pattern met EDA en zij alleen EDA, deze zijn niet direct vergelijkbaar maar leggen wel beide een nadruk op het gebruik van events binnen de flow van het systeem om zo tot een losse koppeling te komen. Wij hebben een beperkt aantal tests gedaan en hebben hiermee slechts een klein aantal datapunten verzameld. Daarom kunnen wij geen conclusies op basis van statistische significantie aan ons onderzoek verbinden. Wel geeft het een inzicht in de complexiteit van de systemen, leercurve en de door ons benodigde tijd voor het doorvoeren van wijzigingen.

Uit het onderzoek bij Sogyo komen twee zaken naar voren. Als eerste blijkt dat het inlezen in een Event Driven Architecture-systeem minder tijd kost vergeleken met het reguliere systeem. Als tweede kwam naar voren dat het implementeren van nieuwe functies juist iets meer tijd kost. Hierbij moet wel opgemerkt worden dat de resultaten tussen beide projecten zeer dicht bij elkaar liggen. Ook is de indicatie die wij hebben op het gebied van de benodigde implementatietijd niet anders dan de conclusie die Sogyo trekt, zij zien tevens geen significant verschil tussen de twee ontwikkelmethoden.

Om een statistisch sterkere vergelijking te kunnen maken, is het beter om soortgelijke datasets als

⁶ http://www.nljug.org/pages/events/content/jfall_2010/sessions/00017/

Sogyo te gebruiken. Deze kleine datasets geven een nauwkeuriger zicht op de ontwikkeltijd. Omdat het kleinere sets zijn is het ook eenvoudiger om meer datapunten te verzamelen. Daarnaast zijn deze sets ook makkelijker af te bakenen, op deze manier is het mogelijk om de overhead als configuratie uit te bannen. Deze punten zorgen voor een statistisch zuiverder experiment. Wanneer er meer zicht nodig is op de ontwikkeling in een realistische setting, is het beter om grotere datasets te hanteren. Zo komt de ontwikkelaar beter in aanraking met de randzaken, die in een echt project natuurlijk ook voorkomen.

Om een betere vergelijking te maken is het onderzoek van Sogyo leidend. Op de manier die zij aanhouden is het resultaat statisch verantwoord dan ons onderzoek. Het is dus van belang om een volgende keer kleinere datasets aan te houden en meer datapunten te verzamelen.

Concluderend, kunnen we stellen dat onze indicaties enigszins bijval krijgen van een vergelijkbaar onderzoek naar een pattern dat raakvlakken heeft met het door ons onderzochte CQRS. Beide onderzoeken concluderen dat er diepgaander en uitgebreider onderzoek nodig is om tot een sterkere conclusie te komen. Het feit dat de gevonden resultaten enigszins overeenkomsten vertonen maakt het gevonden resultaat iets sterker.

5. Ervaringen ontwikkeling

Als onderdeel van ons project hebben we met verschillende third party libraries, tools en frameworks gewerkt. Voorbeelden hiervan zijn onder andere Maven en het Spring framework. Met onze beperkte ervaring op het vlak van Java-ontwikkeling met dergelijke frameworks en libraries zijn we meerdere zaken tegengekomen die de ontwikkeling in meer of mindere mate gehinderd hebben.

We zullen hierbij als eerste kijken naar onze ervaringen met het gebruik van Maven in het algemeen en op het interne netwerk van E.Novation. Vervolgens gaan we over naar dezelfde punten die meer betrekking hebben met het Spring-framework.

5.1. Maven dependency management

De projecten die we als uitgangspunt voor ons onderzoek gekozen hadden maken beide gebruik van Maven⁷ voor het beheren van dependencies. Binnen E.Novation wordt er gewerkt met een lokale repository door middel van Nexus waardoor er beheerd kan worden welke versies van welke libraries e.d. geïnstalleerd mogen worden.

Hierbij bleek na enig speurwerk aan de hand van de ietwat cryptische foutmeldingen van Maven dat sommige door de projecten gebruikte dependencies niet aanwezig waren in de interne repository van het bedrijf. Het was vervolgens onmogelijk om via de lokale Nexus-repository de projecten te installeren. Daardoor moest er gekozen worden voor de (tragere) publieke repositories om de projecten te kunnen compileren.

⁷ <http://maven.apache.org/>

5.2. Gebruik van Spring

Binnen de Spring-projecten waar we mee aan de slag zijn gegaan werd veelvuldig gebruik gemaakt van de 'Autowiring'-functionaliteit van het Spring-framework. Dit houdt in dat er door middel van een annotatie in de code (`@Autowired`) aangegeven kan worden dat bepaalde attributen automatisch ingesteld moeten worden aan de hand van een globale configuratie.

Het probleem hierbij is de beperkte transparantie van deze manier van configuratie. Namelijk, het koppelen van de annotaties en gedefinieerde 'Java Beans' wordt achter de schermen door het framework afgehandeld. Het is meer dan eens voorgekomen dat na het toevoegen van een nieuwe Bean of Autowired-attribuut aan een klasse het systeem op andere plaatsen fouten begon te vertonen.

De bij ons aanwezige onduidelijkheid over de exacte werking van het achterliggende systeem en de beperkte mogelijkheden tot debuggen, maakt het tot een flink vertragsrisico voor de beginnende ontwikkelaar.

5.3. Object relational mappers

Het LA-project maakte in eerste instantie gebruik van het EclipseLink⁸ LA-project om objecten op relationele wijze op te kunnen slaan in de database. Tijdens het schrijven van de integratietests van het systeem bleek hierbij echter onvoorspelbaar gedrag op te treden bij het opzetten en beheren van relaties tussen objecten. Om dit te omzeilen hebben we vervolgens gekozen om met Hibernate 4 te werken, deze vertoont dit gedrag niet.

5.4. Conclusie

Van de voorgaande besproken punten hebben we enige hinder ondervonden in de ontwikkeling. Bij deze factoren zaten we onderaan in de leercurve, met dit project is dat goed gegroeid. Waarvan de Spring wiring voor het meeste oponthoud heeft gezorgd. De beide andere punten waren eenvoudiger. Al met al hebben deze punten enige ruis in de vergelijking gebracht, wat de vergelijking vervuult. Dit en meer zullen we in Aanbevelingen behandelen.

⁸ <http://www.eclipse.org/eclipselink/>

6. Aanbevelingen

Tijdens ons onderzoek waarin we twee projecten wilden op een aantal punten wilden vergelijken zijn we tot een aantal aanbevelingen voor toekomstige onderzoeken gekomen. Deze zullen we kort toelichten. De twee systemen

Een eerste aanbeveling is dat bij een vervolgonderzoek gezorgd moet worden voor stabiele applicaties als uitgangspunt. Het is aan te bevelen om zo min mogelijk ruis te introduceren in de metingen. Wij zijn aan de slag gegaan met twee applicaties die niet volledig stabiel waren voordat we wijzigingen aan de systemen door hebben gevoerd tijdens de experimenten. Hierdoor hebben fouten in het systeem, de configuratie of andere zaken de ontwikkeling verhinderd. De fouten moesten eerst rechtgezet worden voordat de ontwikkeling hervat kon worden. Hiermee zal, ondanks de zorgvuldigheid van de metingen, toch enige ruis in de metingen zijn ontstaan.

Daarnaast, wanneer er significante resultaten gewenst zijn, zal er gezorgd moeten worden voor voldoende tijd om een voldoende aantal metingen te verkrijgen. In het project heeft de nadruk minder gelegen op het doen van een uitgebreid onderzoek naar de verschillen in tijd en meer op documenteren van ervaringen. Hierdoor hebben we geen ruimte gehad om een groot aantal features onafhankelijk door meerdere ontwikkelaars te laten implementeren. Daar zouden we mogelijke metingen op kunnen doen van een hoger betrouwbaarheidsniveau. Wanneer dit wel op deze wijze aangepakt zou worden kunnen er sterkere conclusies getrokken worden uit de verkregen data. Mits er voldoende data verzameld wordt en er met voldoende ontwikkelaars gewerkt wordt.

Een derde punt is de aanbeveling dat er voldoende relevante features gekozen zouden moeten worden om te implementeren. Zo heeft de feature die we aan het begin van het traject bedacht hadden, het toevoegen van een zoekfunctie op voor- en achternaam, weinig impact op het CQRS/ES-deel van de applicaties. Daardoor leverde het weinig inzichten in het gebruik van CQRS. Ter kanttekening heeft deze feature wel voor enige ervaring met het gebruik van MongoDB als query-database gezorgd waardoor dit geen verloren tijd is geweest. In een vervolgonderzoek zou echter voor features gekozen moeten worden met meer relevantie. Daardoor kan er een beter beeld gevormd worden van het verschil tussen de twee systemen.

Een andere aanbeveling is om zoveel mogelijk het te gebruiken design pattern en de te ontwikkelen 'proof of concept'-applicatie op elkaar af te stemmen. Zo bleek tijdens de ontwikkeling van de adresboekapplicatie dat CQRS matig geschikt is voor een applicatie met hoofdzakelijk CRUD-acties. In de praktijk zal dit pattern niet de meest geschikte kandidaat zijn voor deze applicatie. Door de proof of conceptapplicatie af te stemmen met mogelijk toekomstig gebruikt kunnen eventuele bevindingen eenvoudiger doorgetrokken worden.

Tot slot is het aan te bevelen om met een volgend project te zorgen dat er enkel dependencies in Maven ingesteld worden die tevens via Nexus beschikbaar zijn binnen het bedrijfsnetwerk. Op die manier kan er meteen zonder configuratie aan te hoeven passen gewerkt worden met de applicaties, wat het eenvoudiger maakt om aan de slag te gaan met de applicatie.

7. Conclusie

Na de behandeling van de verschillende facetten die in dit onderzoek aan bod zijn gekomen, kunnen we tot slot de balans opmaken. Hierbij is gekeken naar de mogelijke verschillen in ontwikkeltijd, vervolgens complexiteit waarna het Sogyo onderzoek aan bod komt. Afsluitend de waarde van de resultaten.

Het onderzoek naar de benodigde ontwikkeltijd van aanpassingen aan beide projecten heeft geen significante verschillen opgeleverd. Wel geven de resultaten een indicatie dat de extra tijd die gestoken moet worden in het ontwikkelen voor een CQRS-systeem voornamelijk in de tijd die het inlezen in de systemen is gaan zitten. De verwachting is dan ook dat het hier voornamelijk een eenmalige investering betreft die naar mate de ontwikkelaar meer bekend met de werkwijze wordt zal verdwijnen. Naast deze inwerktijd blijkt de tijd die de implementatie heeft gekost bij beide features en systemen nagenoeg gelijk zijn.

Uit de complexiteitsvergelijking van de in het onderzoek uitgebreide systemen zijn enkele bevindingen gekomen. Voor beide projecten blijkt de algehele complexiteit van methoden en klassen hoofdzakelijk laag te zijn, wat te verwachten is voor applicaties met een beperkte featureset, de algehele complexiteit van klassen kent wel enige uitschieters, welke veroorzaakt worden door de relatief lange controller-klassen gebruikt voor de webapplicatie.

Naast de algemene punten zijn er ook nog enkele andere zaken die in het oog springen, zo had de CQRS/ES-gebaseerde applicatie een significant hoger aantal klassen en methoden maar lag de gemiddelde complexiteit hiervan wel lager. Dit duidt op een zekere hoeveelheid overhead die het pattern en het Axon Framework met zich meebrengen en de trade-off tussen de splitsing van concerns, wat kan resulteren op een eenvoudig testbare delen en een compactere algehele applicatie.

Het vergelijkbare onderzoek wat Sogyo heeft laten uitvoeren trekt conclusies die gedeeltelijk overeenkomen met de conclusies wij uit de beperkte set data hebben kunnen trekken. Zo is uit het onderzoek door Sogyo tevens geen significant verschil in ontwikkeltijd en code-complexiteit gebleken wanneer er gebruik gemaakt wordt met het deels vergelijkbare EDA pattern. Dit geeft onze bevindingen een iets sterke ondergrond, al zijn beide onderzoeken niet op gelijke wijze uitgevoerd en daarmee niet direct vergelijkbaar.

Dat brengt gelijk ook het punt met zich mee dat ons onderzoek enige verbeterpunten kent welke besproken zijn in het hoofdstuk 'Aanbevelingen'. Zo zijn onze metingen in een zeer klein aantal gedaan en zijn deze niet geheel ruisvrij geweest door problemen met de gebruikte libraries en tooling van de twee projecten. Bovendien bleken de gekozen features niet geheel representatief voor een vergelijking van de gekozen patterns. Wel heeft het project voor de nodige gedocumenteerde ervaringen met het CQRS/ES-pattern in combinatie met een non relationele database gezorgd en enkel aanwijzingen om voor toekomstige projecten de afweging tussen beide systeemontwerpen te komen.

Bijlage A: Tijdmetingen

Zoekfunctie	Layered Architecture (LA)	CQRS/ES
Gezamenlijk deel	13 uur	
Projectspecifiek deel:		
Inlezen in het huidige systeem	1 uur	7 uur
Debuggen wiring	-	15 uur
Implementatie	6 uur	7 uur
Totaal	7 uur	29 uur

Tabel 1: Gespendeerde tijd voor het toevoegen van de zoekfunctie.

Telefoonnummers	Layered Architecture (LA)	CQRS/ES
Gezamenlijke code	30 uur	
Projectspecifiek deel		
Inlezen systeem	5 uur	3 uur
Implementatie	8 uur	7 uur
Migratie naar Hibernate	3 uur	-
Totaal	16 uur	10 uur

Tabel 2: Gespendeerde tijd voor het doorvoeren van de veranderingen aan het domeinmodel.

Bijlage B: Woordenlijst

Apache Maven - Tool om Javaprojecten te beheren en op automatische wijze te compileren.

Checkstyle - Applicatie voor statische code-analyse welke controleert of Java-code voldoet aan regels omtrent stijl.

CQRS - Command Query Responsibility Segregation.

Command Query Responsibility Segregation - Pattern waarbij de afhandeling van aanpassingen aan het systeem (commands) losgekoppeld is van de presentatie van data.

Cyclomatic complexity - Software metric welke een indicatie geeft van de complexiteit van een stuk software op basis van de hoeveelheid verschillende berekenpaden in het systeem.

Design pattern - Een patroon om een softwaresysteem in te delen om daarmee een veelvoorkomend probleem op te lossen.

EDA - Event Driven Architecture

Event Driven Architecture - Een design pattern gericht op het gebruik van events om de communicatie tussen systeemonderdelen te verzorgen.

Event Sourcing - Pattern waarbij objecten niet in huidige staat worden opgeslagen in een databasesysteem maar enkel een serie events wordt opgeslagen welke, na het opnieuw afvuren van de events, tot het object in de huidige staat leiden.

JavaBeans - Herbruikbare softwarecomponenten voor Java welke voldoen aan een reeks voorwaarden om zo via een consistent interface hergebruikt te kunnen worden binnen applicaties.

LA - Layered Architecture.

Layered Architecture - Een design pattern voor software waarbij er meerdere abstractielagen binnen een applicatie worden ingebouwd waarmee

MongoDB - Een open source document-georiënteerde database.

Nexus - Systeem om Maven repositories mee te beheren en een lokale kopie van software van derden aan te bieden op gecontroleerde wijze.

pmd - Applicatie voor statische code-analyse welke Java-code controleert op potentiële problemen.

Sonar - Applicatie welke verschillende code-analyse applicaties gebruikt om de kwaliteit van code op basis van verschillende metrieken te monitoren.

Spring MVC - Een open source framework om web-applicaties te bouwen.

Bijlage F: Gebruik van Axon en MongoDB

Bachelorproject 2012

Voor de Bachelor Technische Informatica, TU Delft

Maarten van Meijeren, Yorick Holkamp

In samenwerking met *E.Novation*

Begeleiders

Bastiaan Bakker (E.Novation, afdeling Application Development)

Jan Hidders (TU Delft, afdeling Web Information Systems)



Samenvatting

In dit document gaan we in op de werking van 'Command Query Responsibility Segregation'-pattern, Event Sourcing en de combinatie van beide in de vorm van Axon. Daarbij ook de rol van MongoDB binnen Axon. Waarbij bij alle onderwerpen de sterke en zwakke punten belicht worden. Te beginnen met de werking van CQRS en ES.

Het CQRS-pattern zorgt dat de verantwoordelijkheid van het uitvoeren van commands en queries gescheiden van elkaar moet worden afgehandeld. Met Event Sourcing is het mogelijk de historie van wijzigingen van objecten te bewaren. Axon biedt een combinatie van CQRS, (optioneel) ES en een Event Driven Architecture waarmee een applicatie opgezet kan worden.

Het theoretisch model van CQRS heeft zeker voordelen, dat geldt ook voor Event Sourcing. Voor het Axon Framework hoeft dat niet direct te gelden. In bepaalde situaties wanneer de overhead niet hinderlijk is en het event sourcing principe relevant is, is het handig om Axon te gebruiken. Ook is het zaak te letten op de fouten, zoals foutieve event handling, die door de standaardinstellingen van Axon voorkomen.

Niet alle applicaties zijn daar geschikt voor zoals CRUD, write-heavy en realtime applicaties. Applicatie die bijvoorbeeld auditing nodig hebben zijn veel beter geschikt voor ES. CQRS is handig voor grote complexe systemen, waarbij de splitsing en het gebruik van events voor een beter overzicht en eenvoudiger uitbreiding zorgt.

Het gebruik van MongoDB heeft geen significante nadelen tegenover het gebruik van een RDBMS, de grootste drempel voor het gebruik hiervan zal de omscholing van de ontwikkelaars zijn. MongoDB als event store gebruiken kan prima. Met aanpassingen in het Axon Framework is het mogelijk een betere, efficiëntere view-database te maken. De grote winst zit in het gebruik bij de query-database, het is schemaless waardoor het mogelijk is zonder complexe vertaalslag data uit de applicatie direct in de database op te slaan.

1 Inhoud

Samenvatting	2
1 Inhoud	3
2 Inleiding.....	4
3 Werking CQRS/ES.....	5
3.1 CQRS.....	5
3.2 Event Sourcing	5
3.3 Axon Framework	5
4 Afweging CQRS/ES	7
4.1 Passende applicaties	7
4.2 Geen baat bij CQRS/ES.....	8
4.3 Conclusie	8
5 Aandachtspunten Axon Framework	9
5.1 Onafgehandelde events	9
5.2 Foutafhandeling in event handlers	9
5.3 Conclusie	10
6 MongoDB	10
6.1 Event store	10
6.2 Query-database.....	11
6.3 Generieke punten	12
7 Conclusie	12

2 Inleiding

In het onderzoeksproject zijn er verschillende technieken gebruikt om de applicaties te bouwen en te vergelijken. Waaronder CQRS/ES in de vorm van Axon en MongoDB. Om deze twee onderwerpen draait de hele vergelijking in het onderzoek. MongoDB en Axon werken volgens bepaalde principes, echter zitten daar voor- en nadelen aan. In dit document behandelen we de voor- en nadelen van de gebruikte database en design pattern. We bekijken de toepasbaarheid van een combinatie van Axon en MongoDB. De informatie die we hierbij inwinnen is interessant voor E.Novation.

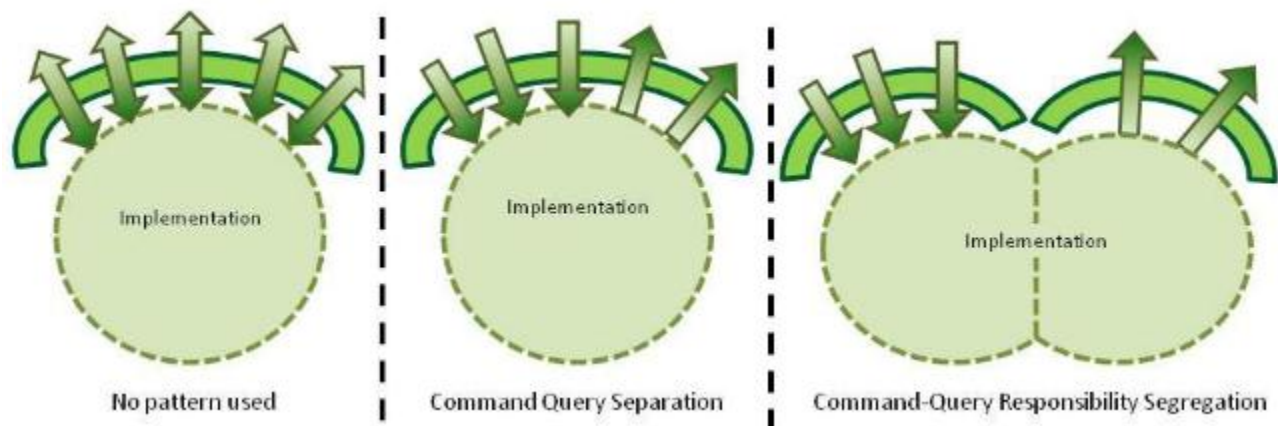
Om een goed beeld te kunnen krijgen van de combinatie van CQRS/ES en MongoDB, behandelen we eerst het principe van CQRS en ES apart, vervolgens samen in het gedeelte over Axon. Vervolgens de toepasbaarheid en de aandachtspunten. Daarna zullen we ingaan op de rol van MongoDB in Axon.

3 Werking CQRS/ES

Om de toepasbaarheid van CQRS/ES en MongoDB te analyseren zal eerst kort de werking van CQRS, Event Sourcing en het Axon Framework besproken worden. We hebben eerder een korte introductie van het CQRS-pattern gegeven in hoofdstuk 3 'CQRS' van het Oriëntatieverslag, dit is een uitgebreide uitleg van dit principe.

3.1 CQRS

Het 'Command Query Responsibility Segregation'-pattern schrijft voor dat de verantwoordelijkheid van het uitvoeren van commands en queries gescheiden van elkaar moet worden afgehandeld. CQRS gaat hiermee, zoals in figuur 1 te zien is, verder op het principe van het Command Query Separation-pattern. Dit pattern schrijft voor dat er voor de acties die de data aanpassen en de acties die data opvragen gescheiden, losse methoden gebruikt moeten worden. Om zo te zorgen dat data zonder neveneffecten opgevraagd moet kunnen worden. Dit heeft voordelen bij het gebruik van bijvoorbeeld assertions in combinatie met het 'design by contract'-principe.



Figuur 1: Overzicht verschillende patterns.

3.2 Event Sourcing

Event Sourcing is een pattern waarmee de historie van wijzigingen van objecten bewaard kan blijven. Dit wordt gedaan door de objecten in de database op te slaan als een sequentie van events. Wanneer vervolgens een object opgevraagd wordt, worden deze events na elkaar toegepast op een 'kaal' object om zo weer tot een object in de huidige staat te komen.

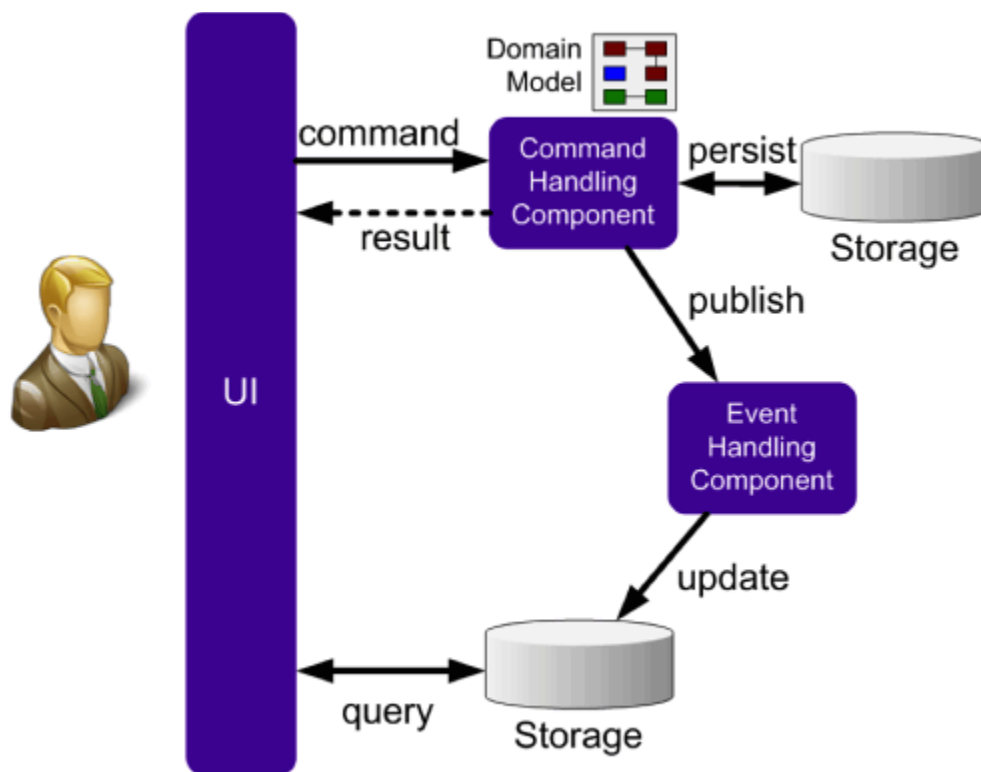
3.3 Axon Framework

Binnen E.Novation wordt een framework gebruikt welke de implementatie van CQRS/ES faciliteert, het 'Axon Framework'. Dit framework biedt een infrastructuur aan om een applicatie op te zetten die het CQRS-pattern volgt. In de praktijk blijkt echter dat Axon veel verder gaat dan het CQRS-pattern an sich stelt. Zo biedt Axon een combinatie van CQRS, (optioneel) Event Sourcing en een Event Driven Architecture waarmee een applicatie opgezet kan worden. Ook wordt de benodigde infrastructuur

gegevens om commands en events af te leveren bij de juiste klassen en objecten eventueel op 'event sourced' wijze op te slaan.

Deze onduidelijkheid in terminologie vanuit Axon zorgt in de praktijk voor enige verwarring. Doordat Axon de gehele samengestelde opzet onder de noemer CQRS schaart terwijl algemene bronnen eerder de losse delen hiervan zullen bespreken. Dit maakt het vooral wat verwarrend aan het begin van het leerproces wanneer bijvoorbeeld zowel de handleiding van Axon als een artikel van bijvoorbeeld Martin Fowler¹ bekeken worden.

Concreet betekent de andere aanpak dat een applicatie opgebouwd met het Axon Framework er een andere structuur op na zal houden dan een applicatie waar enkel het CQRS-pattern in verwerkt zit. Dat kan een enigszins vertekend beeld geven. In een applicatie waar enkel het CQRS-pattern geïmplementeerd wordt zal, zoals eerder besproken, enkel voor scheiding van de afhandeling van de command- en query-opdrachten gezorgd worden. Wanneer echter gekozen wordt om het Axon Framework te gebruiken en de aangeraden opzet gebruikt wordt, zal de applicatie een structuur volgen die meer lijkt op figuur 2.



Figuur 2: Systeemopzet voor een applicatie met het Axon Framework.

¹ <http://martinfowler.com/bliki/CQRS.html>

In deze figuur is te zien hoe het systeem er in grote lijnen uit zal komen te zien. Om te beginnen zorgt de UI-laag enkel voor het aanmaken van commands met betrekking tot een van de domeinmodellen, zoals een commando om een contactpersoon toe te voegen. Na het aanmaken van dit command zal deze gepubliceerd worden om (mogelijk asynchroon) door een command handling component verwerkt worden.

Bij de verwerking van het command zal een event aangemaakt en gepubliceerd worden op de event bus. In het geval van een standaardmethode van opslag zal hierbij het domeinmodel aangepast worden en in de nieuwe staat opgeslagen. Als er echter gebruik gemaakt wordt van Event Sourcing zal enkel het aangemaakte event opgeslagen worden bij het domeinmodel.

Vervolgens zal het gepubliceerde event door een event handling component opgepakt worden. Aan de hand van het event zullen de opgeslagen entries in de query-database bijgewerkt worden met de wijzigingen aan het domeinmodel. Wanneer de gebruiker, bij het gebruik van de applicatie, een verzoek om data doet zal vervolgens, nadat al deze stappen verwerkt zijn, deze aangepaste data getoond worden.

Met deze splitsing van opslag en mogelijk asynchrone verwerking van de commands wordt expliciet duidelijk gemaakt dat een gebruiker mogelijk verouderde data te zien zal krijgen. Dit is echter geen expliciet nadeel voor de gekozen systeemarchitectuur. Een gebruiker van een systeem zal, op mogelijke realtime of single-user systemen na, altijd verouderde data te zien krijgen. Dat komt doordat tegelijk met het tonen van een set data, deze zelfde set al door een andere actor aangepast kan zijn.

4 Afweging CQRS/ES

In hoofdstuk 3 is duidelijk geworden hoe de applicaties in grote lijnen opgebouwd worden volgens het CQRS/ES pattern en de implementatie van Axon Framework. Nu kunnen we inzoomen op het CQRS-pattern. Het pattern heeft verschillende eigenschappen die gevolgen hebben voor applicaties die volgens dit principe werken. In dit hoofdstuk zullen we de verschillende aspecten, met zowel voor- en nadelen, bespreken en vervolgens proberen te analyseren welk type applicatie voordeel zou kunnen behalen uit het gebruik van deze combinatie van twee patterns.

4.1 Passende applicaties

Er zijn verschillende soorten applicaties aan te wijzen die voordeel halen uit de karakteristieke werking van het CQRS/ES pattern. Zo zorgt het Event Sourcing-pattern ervoor dat het op enig moment mogelijk is om vast te stellen wat de staat van een object was op enig moment in de tijd. Daarbij bestaat ook nog de mogelijkheid om na te gaan op welke wijze het object in de huidige staat is gekomen. Deze twee punten zijn vooral interessant wanneer het mogelijk moet zijn om auditing op de data in het systeem toe te passen. De controle met betrekking op de correctheid van gegevens kan nu terugvallen op geschiedenis van alle acties die op de data zijn uitgevoerd. Zo kan eenvoudig bepaald worden of alle acties correct en legaal zijn uitgevoerd.

Daarnaast heeft het CQRS/ES pattern een belangrijk voordeel op het gebied van de overzichtelijkheid van het systeem. Om een bepaald aspect van het systeem aan te passen is er in principe geen vergaande systeemkennis nodig om functionaliteit toe te voegen. Dit heeft vooral voordeel als het systeem groot en complex is. Zo is het mogelijk om zonder het hele systeem van binnen en buiten te kennen, dit toch op een goede manier uit te breiden. De ontwikkeling van complexe systemen krijgt met CQRS/ES een heldere structuur.

4.2 Geen baat bij CQRS/ES

Niet alle soorten applicaties profiteren van de werking van CQRS/ES, sommige systemen kunnen CQRS/ES beter vermijden. Zoals hierboven aangegeven profiteert vooral de ontwikkeling van een complexe applicatie daarvan, daar tegenover staan de eenvoudige CRUD applicaties. Dit soort applicaties kan beter geen CQRS/ES gebruiken. De overhead om het pattern toe te passen is dan erg groot, terwijl het nauwelijks positieve invloed heeft de overzichtelijkheid.

Ook voor applicaties die erg veel schrijven naar de database is CQRS/ES niet het optimale pattern. Event Sourcing is namelijk niet erg snel en moet toch alle wijzigingen bijhouden. Dat betekent dat bij elke wijziging de hele reeks aan events opgehaald en weer weggeschreven moet worden. Daarnaast zal bij een erg schrijffintensieve applicatie de Event Store snel ontzettend groot worden.

Tot slot het feit dat CQRS/ES gebruik maakt van twee verschillende soorten databases en daarbij de query database niet altijd up-to-date is zorgt voor een ander nadeel. De applicaties die (semi-)realtime zijn, kunnen uit de query database de data halen. Het kan dan zijn dat de data dan niet up to date is. Dat is voor realtime applicaties een must, daar kan niet aan worden voldaan.

4.3 Conclusie

Samengevat zijn applicaties die geen gebruik maken van de karakteristieke werking van CQRS/ES beter af als deze dit pattern niet gebruiken. Dat zijn onder andere CRUD, write-heavy en realtime applicaties. De applicaties die gebruik maken van een Event Store, voor bijvoorbeeld auditing, zijn veel beter geschikt voor CQRS/ES. Tevens is CQRS handig voor grote complexe systemen, met dit pattern is er een beter overzicht mogelijk en eenvoudiger uitbreiding. CQRS an sich is een goed pattern wat een systeem overzichtelijk kan houden. CQRS/ES in de vorm van Axon gaat veel verder is daardoor complexer om goed toe te passen.

5 Aandachtspunten Axon Framework

De implementatie van Axon Framework kent enkele gevoeligheden, waar bij de toepassing van het framework rekening mee gehouden moet worden. Zo kom je niet voor verrassingen komen te staan als de software in productie is. Hierbij zullen we als eerst kijken naar het gedrag van Axon wanneer er events in de applicatie gebruikt worden zonder bijbehorende event handler. Daarbij de implicaties die dit mogelijk heeft voor een applicatie. Tot slot bekijken we kort de methode van foutafhandeling binnen event handlers en het gevolg hiervan op foutopsporing en mogelijk dataverlies.

5.1 Onafgehandelde events

In Axon's standard-EventHandler worden events waar geen event handler voor gevonden kan worden 'stilletjes' zonder logging of exception genegeerd². Dit is een potentieel risico tijdens de ontwikkeling van een applicatie, wanneer er bijvoorbeeld een annotatie van een event handler methode vergeten wordt zal deze niet gedetecteerd worden. Mocht deze code vervolgens ook echt in productie belanden, omdat er bijvoorbeeld een gebrekkige testprocedure is, dan zullen er events zijn die niet afgehandeld worden. Het gevolg is dat de staat van de query-database niet meer overeen komt met die van de domeinmodellen.

Om dit gedrag te vermijden kan de ontwikkelaar zelf een abstracte klasse toevoegen aan het systeem en deze als superklasse instellen voor klassen met event handling functies. Wanneer deze abstracte klasse dan een functie bevat welke het meest generieke type event accepteert, zal deze methode uitgevoerd worden als er geen, minstens zo specifieke, methode voor handen is. In de implementatie van deze methode kan vervolgens een melding naar een log weggeschreven worden zodat dit soort fouten opgespoord kunnen worden.

5.2 Foutafhandeling in event handlers

Het standaardgedrag van Axon bij het optreden van een exception tijdens het draaien van een event handler op een bepaald event is vergelijkbaar met de behandeling van events die niet worden afgehandeld. Wanneer er bij de afhandeling van een event een exceptie optreedt, bij bijvoorbeeld een databaseservice die niet bereikbaar is of een model dat niet gevonden kan worden, zal Axon met de normale configuratie geen actie ondernemen en enkel dit event stilletjes overslaan³.

Door het gedrag van de event scheduler aan te passen kan er gezorgd worden dat de events die mislukken, door bijvoorbeeld fouten in de event handler, nogmaals uitgevoerd worden. De event handler kan vervolgens uitgebreid worden met een controle. In het geval van een 'retry' kan deze controle een bericht naar een log te schrijven zodat dit gedrag ook daadwerkelijk op te sporen is. Zo kan er door bijvoorbeeld een applicatiebeheerder naar gekeken kan worden.

² <http://www.axonframework.org/apidocs/1.3/org/axonframework/eventhandling/annotation/EventHandler.html>

³ <http://www.axonframework.org/docs/1.3/event-processing.html#d4e975>

5.3 Conclusie

Het gebruik van het Axon Framework het leven van een developer zeker kan vereenvoudigen bij de implementatie van een CQRS/ES-systeem. Echter kent dit framework ook enkele valkuilen in de standaardinstellingen. Hierdoor is het nodig om als ontwikkelaar de gehele documentatie zorgvuldig door te nemen. Daarnaast is het mogelijk om een ervaren Axon ontwikkelaar naar het systeem te laten kijken om op de mogelijke valkuilen van het systeem te wijzen.

6 MongoDB

Naast de toepasbaarheid van CQRS/ES is het ook interessant om een blik te werpen op het gebruik van een NoSQL-databasesysteem. NoSQL zou voordelen met zich mee kunnen brengen. In het eerdere document 'Research Document' is hierbij aan de hand van voor E.Novation relevante kwaliteitsattributen gekozen om de focus te leggen op het gebruik van MongoDB. De specifieke voor- en nadelen van dit systeem zijn tevens terug te vinden in het document waaraan gerefereerd is.

In het CQRS/ES systeem (het Axon Framework) dat we in ons onderzoek hebben gebruikt bestaat standaard de mogelijkheid om MongoDB te gebruiken als view database en event store. Voor deze specifieke functie zijn er speciale eigenschappen van MongoDB belangrijk of zouden belangrijk kunnen zijn. In dit hoofdstuk zullen we ingaan op de voor- en nadelen van deze keuze. Om tot slot te concluderen in hoeverre dit zich doorvertaalt naar voordelen voor het bedrijf. Om dit te kunnen doen zullen we eerst kort ingaan op het gebruik van MongoDB als event store en view database om vervolgens enkele algemene voor- en nadelen te bespreken waarna we tot een conclusie over zullen gaan.

6.1 Event store

Als het Axon Framework ingesteld staat met het gebruik van 'event sourcing', worden alle events die op de verschillende domeinmodellen uitgevoerd worden stuk voor stuk opgeslagen. Wanneer vervolgens een nieuwe actie toegepast moet worden kan er door al deze events na elkaar uit te voeren op een nieuw 'kaal' object de uiteindelijke staat teruggehaald worden.

In de huidige implementatie van het Axon Framework is ondersteuning voor het gebruik van MongoDB als opslagmedium. Deze implementatie maakt echter geen gebruik van de mogelijkheden die MongoDB biedt ten opzichte van relationele databases waardoor er geen duidelijke voordelen aan het licht komen bij de keuze voor MongoDB als opslagmedium.

6.1.1 Mogelijke verbeterpunten

De standaardimplementatie van Axon slaat deze events op in een collectie als document met de velden 'id', 'aggregateld', 'model', 'serializedEvent', een timestamp en sequence number. Hierbij wordt een grote hoeveelheid documenten aangemaakt. Als een bepaald domeinmodel met een bepaald 'aggregateld' op gehaald moet worden, moeten alle events met dit id opgehaald worden uit de database.

Deze aanpak is momenteel identiek aan de methode gehanteerd voor de opslagmethoden in een relationele database en biedt ruimte voor verbetering. Zo zou het mogelijk zijn om de events op te slaan in één document per domeinobject, waarbij dit document zou kunnen volstaan met de velden 'aggregateld', 'serializedEvents' en 'type' waarna 'serializedEvents' een array van events zou kunnen bevatten. Door deze aanpassing door te voeren zouden alle events van een domeinobject met slechts één query opgehaald kunnen worden wat voor enige snelheidswinst zou kunnen zorgen. Door vervolgens ook te werken met slechts één identifier per domeinmodel, namelijk het id van dit model, zou er tevens bespaard kunnen worden op de benodigde ruimte voor een index op de collectie. Dat komt namelijk doordat er geen index op zowel id als aggregateld meer nodig is.

Een andere mogelijke verbetering is het benutten van de mogelijkheid van MongoDB om met JSON te werken. Deze feature staat op de planning voor de release van Axon Framework 2.0⁴. Momenteel wordt door het Axon Framework enkel gebruik gemaakt van XML, om events naar XML te converteren voor opslag. Wanneer hierbij gekozen wordt om in plaats van XML gebruik te maken van JSON wordt enige overhead in de opslag overgeslagen. Bovendien kan er gebruik gemaakt worden van de query-syntax van MongoDB om de events in de database te doorzoeken. Dit laatste kan vooral interessant zijn om analyses van de frequentie van voorkomen van events te maken.

6.2 Query-database

De query-database bevat de data die wordt gepresenteerd in de applicatie. Aan de hand van de acties van een gebruiker kunnen deze geraadpleegd worden. Bij het gebruik van MongoDB als view database komt het voordeel vooral bij de schemaless aanpak van dit databasesysteem naar voren. Dit heeft als voordeel dat de opslag van data één op één overeen kan komen met het datamodel wat gehanteerd wordt. Zo kan er, om gegevens aan de gebruiker te tonen, gewerkt worden met eenvoudige queries. Er is nu geen noodzaak om verschillende tabellen te combineren wat de ontwikkeling kan versimpelen en de performance kan verhogen. Er is dus alle vrijheid om de invulling van de database te bepalen zonder dat er rekening gehouden moet worden met het verschil tussen de manier van datarepresentatie in de applicatie en de beperkingen van het opslagmedium.

Een ander voordeel is dat de datamigratie in bepaalde gevallen mogelijk eenvoudiger uit te voeren is. Als er geen vast databaseschema aanwezig is in de database kunnen nieuwe velden toegevoegd worden door deze simpelweg naar de database te sturen. Dat het on-the-fly aangepast kan worden biedt mogelijk voordelen op systemen waarin geen of zo weinig mogelijk downtime is gewenst. Zo heeft het toevoegen van een veld aan een tabel met veel entries geen enkele invloed op de performance van het systeem. Op overige vlakken als performance en featureset, welke ook relevant zijn voor een query-database, zijn er verder geen significante algemene verschillen te ontdekken tussen MongoDB en een relationele database.

Het antwoord op de vraag of er winst te behalen is door MongoDB te gebruiken voor een query-database zit hem dan ook voornamelijk in de ontbrekende vertaalslag tussen de applicatielaag en de

⁴ <https://github.com/AxonFramework/AxonFramework/commits/master/mongo/src/main/java/org/axonframework/serializer/bson/DDBObjectXStreamSerializer.java>

uiteindelijke opslag. Dat kan voor eenvoudigere code zorgen, het gebruik van een zwaar ORM is namelijk niet nodig. Ook de mogelijkheid om zonder joins 1:n-relaties in een enkel document te kunnen representeren is een reden. Tot slot de mogelijke eenvoudigere aanpassingen aan het databaseschema en bijbehorende impact op de beschikbaarheid van het systeem.

6.3 Generieke punten

Naast de punten met betrekking tot de werking van de twee specifieke toepassingen zijn er ook nog algemene zaken bij het gebruik van MongoDB komen kijken.

Wanneer er gekozen wordt voor het gebruik van MongoDB binnen een project zullen niet alle ontwikkelaars al ervaring hebben met het gebruik van dit systeem. Dat kan betekenen dat de ontwikkelaars mogelijk eerst door een leerproces zullen moeten. Daarmee zal de ontwikkeling in eerste instantie langzamer verlopen. Gelukkig kent MongoDB, zoals in het research document is besproken, een sterke set documentatie op de website van het systeem. Daarnaast is de leercurve van het gebruik van het systeem relatief beperkt doordat de meeste interactie met de database op gelijkwaardige wijze verloopt.

Ditzelfde geldt ook voor het onderhoud van de applicaties. Wanneer de applicaties eenmaal opgeleverd zijn zullen systeem- en applicatiebeheerders waarschijnlijk nog niet bekend zijn met deze nieuwe programmatuur. Dit vereist inspanning om de kennis op niveau te brengen. Ook kent MongoDB een minder grote hoeveelheid aan beschikbare tooling dan een ouder project zoals PostgreSQL wat mogelijk implicaties voor beheer met zich meebrengt.

7 Conclusie

Het theoretisch model van CQRS heeft zeker voordelen, dat geldt ook voor Event Sourcing. Voor het Axon Framework hoeft dat niet direct te gelden. Alleen in bepaalde situaties wanneer de overhead van het framework op de koop toegenomen kan worden en het event sourcing principe relevant is, is het handig om Axon te gebruiken. MongoDB gebruiken als event store kan prima, er zijn mogelijkheden om met aanpassingen in het Axon Framework een betere, efficiëntere view-database te maken. Mogelijk komt dit beschikbaar in volgende versies.

De grote winst zit in het gebruik bij de query-database, het is schemaless daardoor is er geen zware ORM nodig om de vertaling te maken tussen de applicatie en de database. Daartegenover staat de omschakeling van de ontwikkelaars, echter hoeft dat probleem niet groot te zijn. De syntax is eenvoudig en er zijn prima drivers beschikbaar voor Java.