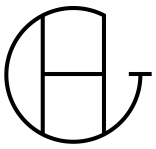


HoGent

Faculteit Bedrijf en Organisatie

Onderzoek naar de meerwaarde van EventSourcing als gegevensbeheertechniek bij Skedify, waar ze gespecialiseerd zijn in online afspraak beheersoftware.

Robin Malfait

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Lieven Smits
Co-promotor:
Mathias Verraes

Instelling: Skedify NV

Academiejaar: 2016-2017

Tweede examenperiode

Faculteit Bedrijf en Organisatie

Onderzoek naar de meerwaarde van EventSourcing als gegevensbeheertechniek bij Skedify, waar ze gespecialiseerd zijn in online afspraak beheersoftware.

Robin Malfait

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Lieven Smits
Co-promotor:
Mathias Verraes

Instelling: Skedify NV

Academiejaar: 2016-2017

Tweede examenperiode

Samenvatting

De maatschappij draait om data. Data zijn de gegevens die gepersisteerd zijn in een databank. Informatie is een afgeleide van deze data. Data en informatie zijn geld waard. Daarom is het van belang dat er voorzichtig met data wordt omgegaan zodat er geen gegevens verloren gaan bij mutaties in het systeem. In systemen waar Relational Database Management Systemś (RDBMSs) gebruikt worden, gaat er data verloren wanneer er niet genoeg nagedacht wordt. Bij elke wijziging of verwijdering van een rij, gaat de voorgaande informatie die ter beschikking was verloren. Het probleem hierbij is dat deze databank gebruikt wordt als single source of truth. Dit kan echter opgelost worden door Logs te gebruiken, maar vermits deze niet de single source of truth zijn, en er geen huidige staat van berekend wordt, is de kans groter dat deze fouten bevat of onbruikbaar is om de oude gegevens te achterhalen. Het kan ook zijn dat er gegevens niet gelogd werden door een vergeetachtigheid van een developer. Door gebruik te maken van EventSourcing gaat er geen data verloren. Dit komt omdat EventSourcing gebruik maakt van een EventStore (zie Hoofdstuk 5.7) als single source of truth waar de huidige staat van berekend werd. Elke actie die gebeurt wordt eerst gepersisteerd in de EventStore, en daarna worden de projecties (zie Hoofdstuk 5.9) pas uitgevoerd.

Het is van belang om informatie bij te houden, dit kan altijd interessant zijn in de toekomst wanneer er specifieke businessvragen komen waar er momenteel nog geen antwoord op is.

Eerst wordt er een literatuurstudie gedaan omtrent EventSourcing en zijn voor- en nadelen. Daarna wordt er gekeken naar alles wat er nodig is om aan een EventSourcing applicatie te beginnen. Dit gaat van Command Query Separation (CQS), Command Query Responsibility Segregation (CQRS) tot de effectieve onderdelen van EventSourcing. Er wordt ook uitleg gegeven over Skedify zodat de concrete businesscase duidelijk gescoped wordt.

Voorwoord

Ik ben al een 5 jaar lang gepassioneerd door data en informatie. Door gebruik te maken van EventSourcing gaat er geen informatie verloren, en is er een mogelijkheid om te tijdreizen. Het is mogelijk om een rapport te schrijven in het heden, terug te gaan naar het verleden en daar het rapport te genereren alsof het rapport toen al bestond. Het idee van deze bachelorproef kwam uit gesprekken met een aantal mensen via de sociale media Twitter. Deze mensen zijn nu bekend in Domain Driven Design/Development (DDD) en EventSourcing wereld. Een van deze mensen is Shawn McCool, die het platform https://eventsourcery.com heeft gemaakt, wat een introductie is tot domain modeling, CQRS (zie Hoofdstuk 4) en EventSourcing (zie Hoofdstuk 5).

Dankzij Twitter heb ik ook mijn co-promotor, Mathias Verraes, leren kennen. Ik heb hem ook ontmoet op een conferentie in Nederland, namelijk Laracon EU 2014. Ik zou hem heel graag willen bedanken voor het helpen realiseren van deze bachelorproef. Ik kon altijd met al mijn vragen terecht bij hem, en hij bezorgde mij ook de nodige boeken, blog posts en andere resources omtrent EventSourcing. Ik kreeg ook de kans om naar Meetup te gaan, maar dit was niet altijd even gemakkelijk omdat ze niet altijd bij de deur plaatsvonden.

Ik zou ook graag de product owner van Skedify, Christophe Thelen, bedanken om mij interessante gegevens over Skedify te bezorgen, alsook resources in verband met EventSourcing omdat hij hier ook al in de praktijk mee gewerkt heeft.

Daarnaast zou ik ook graag mijn promotor, Lieven Smits, willen bedanken voor het benadrukken van mijn spellingsfouten, mij te helpen bij het herstructureren van bepaalde delen en in het algemeen om deze bachelorproef tot een goed einde te brengen.

Tot slot zou ik ook graag mijn vriendin willen bedanken voor het extra nalezen van mijn bachelorproef en mij te steunen in deze drukke tijden.

Inhoudsopgave

1 Inleiding 11

1.1 Stand van zaken 11

1.2 Probleemstelling en Onderzoeksvragen 11

1.3 Opzet van deze bachelorproef 12

2 Methodologie 13

2.1 Literatuurstudie 14

2.2 Proof of concept 14

3 Skedify 15

4 CQRS 17

5 EventSourcing 21

5.1 Aggregates 22

5.2 Value Objects 22

5.3 Messages 24

5.3.1 Imperative Messages 24

5.3.2 Interrogatory Messages 24

5.3.3 Informational Messages 25

5.4 Domain Events 26

5.5 Invariants 26

5.6 Eventual Consistency 27

5.6.1 Transactional Consistency 27

5.7 Event Store 27

5.8 Audit Log 28

5.9 Projections 29

5.10 Testen 29

5.10.1 Debugging 30

6 Schaalbaarheid 31

7 Voordelen 33

8 Nadelen 35

9 Kosten 37

9.0.1 Infrastructurele kosten 37

9.0.2 Developer kosten 38

9.0.3 Performance kosten 39

10 Proof of concept 41

11 Conclusie 43

Lijst van acroniemen 45

Verklarende woordenlijst 48

12 Bijlage A: Bachelorproefvoorstel 49

Bibliografie 53

1. Inleiding

1.1 Stand van zaken

1.2 Probleemstelling en Onderzoeksvragen

Data is het belangrijkste gegeven van een bedrijf. Rapportering is van belang om de eisen van de klant nog beter te vervullen. Voorspellen van vragen die de business zou kunnen stellen binnen dit en 5 jaar, is de dag van vandaag moeilijk. Daarom is het interessant om alle informatie bij te houden om hier interessante rapportering op te kunnen uitvoeren. In een systeem met een RDBMS wordt niet alle informatie bijgehouden. Bij het verwijderen of wijzigen van informatie is de voorgaande informatie verloren gegaan. Alle informatie bijhouden is dus interessant, maar wat zijn hier de voor- en nadelen van?

De onderzoeksvraag luidt als volgt: Wanneer is EventSourcing een meerwaarde voor een bedrijf zoals Skedify (Hoofdstuk 3), waar ze gespecialiseerd zijn in online afspraakbeheer-software?

Hierbij worden ook volgende subvragen beantwoord:

- Wat is het verschil tussen een systeem met relationele databank modellen en een systeem met EventSourcing?
- Hoe kan een applicatie die gebruik maakt van EventSourcing getest worden?
- Welke delen moeten er EventSourced worden?

1.3 Opzet van deze bachelorproef

De rest van deze bachelorproef is als volgt opgebouwd:

In Hoofdstuk 2 wordt de methodologie toegelicht en worden de gebruikte onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeksvragen.

In Hoofdstuk 11, tenslotte, wordt de conclusie gegeven en wordt een antwoord geformuleerd op de onderzoeksvragen. Daarbij wordt ook een aanzet gegeven voor toekomstig onderzoek binnen dit domein.

2. Methodologie

Voor er een proof of concept gemaakt kan worden in verband met EventSourcing, moet men eerst kijken naar wat EventSourcing is en wat het inhoudt. Naast EventSourcing zijn er ook concepten zoals CQRS die even aan bod moeten komen omdat dit een interessante rol speelt bij EventSourcing. CQRS is een splitsing van de lees- en schrijfkant in een systeem. Binnen EventSourcing bestaan er ook Commands (schrijfkant) en Queries (leeskant), en daarom is dit een goed hoofdstuk om binnen deze bachelorproef op te nemen. Daarnaast draait het in dit onderzoek rond Skedify (Hoofdstuk 3), het bedrijf dat als concrete case van toepassing is voor deze bachelorproef. In het deel rond Skedify, kan er meer te weten gekomen worden over wat ze doen. Daarna zal er gekeken worden naar de huidige manier van werken bij het bedrijf omtrent hun data. Tot slot gaat alles worden samengevoegd: Skedify en de huidige manier van werken, en Skedify en EventSourcing. Hierna worden de voor- en nadelen van beide mogelijkheden afgegaan om te zien of EventSourcing nu effectief een meerwaarde biedt voor Skedify. Het onderzoek is in grote delen opgesplitst: Skedify, EventSourcing, huidige manier van werken en de conclusies.

In het deel over Skedify zal er onderzocht worden wat ze doen, en hoe ze dit doen. Er zal ook een gesprek komen met de businesskant van Skedify om vragen te kunnen beantwoorden en om duidelijke conclusies te kunnen trekken.

In het deel over EventSourcing zal er dieper onderzocht worden hoe EventSourcing juist werkt en hoe het gebruikt kan worden.

2.1 Literatuurstudie

De eerste fase van dit onderzoek start met het vergaren van informatie door middel van een literatuurstudie. Het doel van deze literatuurstudie is om een basisinzicht en kennis te verkrijgen met betrekking tot EventSourcing. Alvorens een antwoord op de onderzoeksvragen kan geformuleerd worden, moet er vanzelfsprekend een basiskennis zijn over de belangrijkste zaken die te maken hebben met EventSourcing. Het is van belang om eerst de basis principes van het CQRS en EventSourcing te begrijpen.

2.2 Proof of concept

Voor de proof-of-concept zullen bepaalde functies van Skedify geschreven worden in het huidige systeem en in een EventSourcing systeem. Daarbij zal er gekeken worden naar voordelen en nadelen van beide systemen. Met de volgende zaken zal rekening worden gehouden:

- Hoe lang werd er gewerkt aan een bepaalde feature?
- Hoe leesbaar/onderhoudbaar is de code?
- Hoe kan een bepaalde feature getest worden?
- Is de feature een 1-op-1 mapping met de business?

3. Skedify

Skedify is een Gentse tech start-up dat online afspraakbeheersoftware ontwikkelt voor onder andere de banksector, ziekenfondsen, verzekeringen, immobiliën en HR-bedrijven. Deze bachelorproef gaat op zoek naar een antwoord of EventSourcing een meerwaarde kan bieden voor Skedify. Skedify zorgt er voor dat een persoon in eender welke sector een afspraak kan maken op een korte tijd voor eender welk probleem, en dit met de juiste contactpersoon van het bedrijf. Dit moet in een zo kort mogelijke tijd verlopen. Skedify is een business-to-business tool, die onder andere dynamisch formulieren kan gaan opbouwen om te gebruiken in de plugin die op de website van de klant terecht komt. Alle informatie binnen het systeem wordt in een relationele databank opgeslagen.

Een van de mogelijkheden die Skedify biedt is het wijzigen van een afspraak. Het kan interessant zijn om te weten hoeveel keer dit gebeurt en of er specifieke maatregelen kunnen genomen worden.

Rapportering is tot op de dag van vandaag nog niet beschikbaar omdat er geen historiek wordt bijgehouden.

4. CQRS

CQRS is een term uitgevonden door Greg Young. Greg Young gaat ook verder in op EventSourcing. Een goede uitleg hieromtrent is te vinden in een presentatie die hij gaf op een conferentie (Young, 2014). CQS, kort voor Command Query Seperation, is een principe die uitgevonden is door Bertrand Meyer (Meyer, 1988). Het is een API design principe dat beschrijft hoe er met een object of een systeem gecommuniceerd moet worden. CQRS daarentegen is een architectuurstijl die gaat over hoe dat aan de binnenkant geïmplementeerd is. Als er gekeken wordt naar CQS is dit al een eerste vorm van goede, overzichtelijke code schrijven. CQS zorgt er voor dat Getters en Setters gescheiden zijn. Getters zijn louter bedoeld om een waarde uit de huidige state te halen en deze terug te geven. Setters zijn bedoeld om een wijziging te doen (of een algemene actie uit te voeren), Setters geven geen waarde terug maar Void, dit principe wordt ook uitgelegd op de blog van Martin Fowler (Fowler, 2005a). Een goede heuristiek voor het toepassen van CQS is: "Een vraag stellen, mag het antwoord niet wijzigen", met andere woorden, een Getter heeft nooit gevolgen op de huidige state of op andere zaken.

Het doel van CQS is vooral de leesbaarheid verhogen, het is een principe die developers gebruiken en begrijpen waardoor communicatie makkelijker gaat. Een tweede probleem is dat Getters en Setters in één en dezelfde klasse gedefinieerd zijn. Voor eenvoudige klassen is dit geen probleem, maar wanneer er gelezen of geschreven wordt naar een databank is dit wel een probleem. Er is geen strikte scheiding tussen de leeskant en de schrijfkant.

Listing 4.1: Voorbeeld van CQS

```
1  class Appointment {
2    private DateTime start;
3    private DateTime end;
4    private String subject;
5
6    public DateTime getStart() {
7      return this.start;
```

```
8     }
9
10    public DateTime getEnd() {
11      return this.end;
12    }
13
14    public String getSubject() {
15      return this.subject;
16    }
17
18    public void setStart(DateTime start) {
19      this.start = start;
20    }
21
22    public void setEnd(DateTime end) {
23      this.end = end;
24    }
25
26    public void setSubject(String subject) {
27      this.subject = subject;
28    }
29  }
```

In dit voorbeeld zijn Getters en Setters strikt gescheiden. Het opvragen van een datum of subject wijzigt niets aan de huidige state van dat object. Wanneer deze klasse ook verbonden is aan een databank, dan zullen er zich problemen vormen, wat te zien is in het volgende voorbeeld.

Listing 4.2: Voorbeeld van CQS met databank

```
1   @Entity
2   @Table(name="appointments")
3   class Appointment {
4     private DateTime start;
5     private DateTime end;
6     private String subject;
7
8     public DateTime getStart() {
9       return this.start;
10    }
11
12    public DateTime getEnd() {
13      return this.end;
14    }
15
16    public String getSubject() {
17      return this.subject;
18    }
19
20    public void setStart(DateTime start) {
21      this.start = start;
22    }
23
24    public void setEnd(DateTime end) {
25      this.end = end;
26    }
27
28    public void setSubject(String subject) {
29      this.subject = subject;
30    }
31  }
```

Er is nu geen manier om de lees- en schrijfkant los te koppelen van elkaar. De Appointment klasse zal zowel voor het persisteren van data met deze databank verbinden als voor het uitlezen van gegevens. Er is momenteel geen manier om te bepalen dat het lezen van data

via een andere database connectie moet gaan. Het lezen en schrijven is gekoppeld aan deze klasse waardoor er geen scheiding mogelijk is.

Dit is waar CQRS, Command Query Responsibility Segregation, komt kijken. CQRS zorgt er voor dat de lees- en schrijfkant kant strikt gescheiden zijn. Het zijn bijna 2 applicaties die naast elkaar staan. Een Command wordt afgehandeld aan de schrijfkant en een Query wordt afgehandeld aan de leeskant. Zowel een Command als een Query zijn messages, dit wordt verder besproken in hoofdstuk 5.3. De leeskant gaat zijn informatie halen bij de databank (of een andere vorm van opslagmechanisme), dit kan via Structured Query Language (SQL) queries, Object Relational Mapper (ORM) tools, enzovoort. De manier waarop dit gebeurt staat volledig los van hoe de schrijfkant communiceert met het opslagmechanisme.

De meeste applicaties, onder andere ook die van Skedify, zijn intensiever aan de leeskant dan aan de schrijfkant (Tabel 4.1). De lees- en schrijfkant zijn nu strikt gescheiden en er kan gebruik gemaakt worden van schaling mechanismen om de performantie te verhogen (zie Hoofdstuk 6). Beter nog, de leeskant en schrijfkant kunnen individueel geschaald worden. Er kan zelfs geopteerd worden om lees- en schrijfkant in verschillende programmeertalen te schrijven.

Tabel 4.1: Aantal requests vergeleken voor de lees- en schrijfkant. Er zijn 5.548 (~18.02%) keer zoveel lees requests dan schrijf requests.

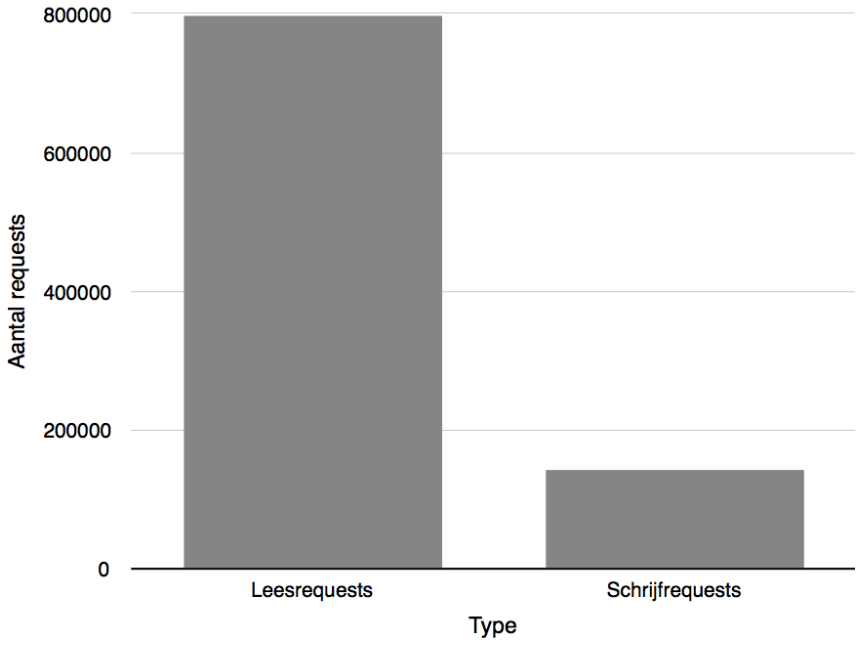
Methode	# Requests	Type
GET	612 000	LEZEN
OPTIONS	148 000	LEZEN
POST	129 000	SCHRIJVEN
HEAD	362 000	LEZEN
PATCH	13 400	SCHRIJVEN
DELETE	1 100	SCHRIJVEN

Door de zwaardere druk langs de leeskant, kan er moeilijk geoptimaliseerd worden voor alle cases waar zowel de lees- als schrijfkant voordeel uit halen. De splitsing is daarom een voordeel om langs beide kanten te kunnen optimaliseren.

Wanneer de lees- en schrijfkant niet gescheiden zouden worden, dan maken ze gebruik van de dezelfde databank. Wanneer er gelezen wordt en door een andere partij geschreven wordt, kan dit elkaar hinderen omdat beide kanten dezelfde resources gebruiken.

CQRS speelt een grote rol bij EventSourcing, vandaar dit korte hoofdstuk.

Figuur 4.1: Visuele representatie van tabel 4.1



5. EventSourcing

EventSourcing is geen nieuwe uitvinding, maar het principe wordt al jaren gebruikt in andere sectoren zoals de bankindustrie, wetgeving, en patiënten fiches die opgeslagen zijn bij dokters. De Belgische wetgeving maakt gebruik van het principe van EventSourcing in die zin dat, bij het toevoegen of ongedaan maken van wetten, er een addendum wordt toegevoegd aan de basiswetgeving om een nieuwe wet in te voeren, of een oude wet ongedaan te maken („Loi - Wet", g.d.).

EventSourcing betekent, zoals de naam al verklapt, dat events de bron zijn van de applicatie. Het is ook zo dat de events de "single source of truth" zijn van de applicatie om andere state uit af te leiden. Een log bestaat ook uit een geschiedenis van events, maar deze wordt niet gebruikt als "single source of truth" op de huidige state af te leiden.

EventSourcing wordt door Microsoft aanzien als een patroon. Microsoft heeft onderzoek gedaan naar wanneer men dit patroon het best gebruikt en wanneer niet (Microsoft, 2017).

Zoals eerder vermeld, is het principe van EventSourcing niet nieuw voor sectoren zoals de bankindustrie, daarom zijn er veel voorbeelden te vinden over bank transacties en e-commerce omtrent EventSourcing. Deze worden ook als voorbeeld gebruikt bij Microsoft (2017). In deze bachelorproef wordt er naar Skedify gekeken, wat niets te maken heeft met de standaard voorbeelden die te vinden zijn.

Als er naar de bankindustrie wordt gekeken dan is het bedrag op iemand zijn rekening niet een getal dat enkel en alleen opgeslagen is in een databank. Er is een lijst van transacties die tot dit getal komen. Het getal op een zichtrekening zit ook opgeslagen in een databank, maar dit is puur als Caching mechanisme, zodat men niet elke keer opnieuw alle transacties moet afgaan om dit getal te bepalen.

Het is zo dat EventSourcing de laatste jaren pas aan het opkomen is in de informatica sector. In de volgende hoofdstukken zal er dieper ingegaan worden op onderdelen van EventSourcing.

5.1 Aggregates

Aggregate is een patroon die in de DDD wereld gebruikt wordt (Fowler, 2005b). Aggregaten zijn de entiteiten die behoren tot het domein. Een aggregaat is een entiteit die zal zorgen dat invariants (besproken in 5.5) gegarandeerd zullen worden. Eens dit het geval is, zal de aggregaat er voor zorgen dat de juiste domain events opgeslagen worden en de state van de applicatie wordt gewijzigd.

Deze aggregaten moeten correct gekozen worden, zodat er geen gevaar is dat er 'God' aggregaten ontstaan. Een 'God' aggregaat is een aggregaat dat alles in een applicatie doet, in de meeste gevallen is dit een 'User' aggregaat.

In het geval van Skedify zijn goede aggregaten: Appointment, Subject, Question, Answer, Office, ... omdat deze core concepts zijn bij Skedify.

5.2 Value Objects

Value objects zijn objecten die zorgen dat een bepaalde waarde altijd in een geldige staat is. Bijvoorbeeld, er kan een value object zijn voor een e-mail. Dit e-mailadres moet ten alle tijde geldig zijn. Dit kan afgedwongen worden door in de constructor van een klasse een waarde te ontvangen en deze te controleren. Indien deze waarde fout is, wordt er een exceptie gegooid. Deze exceptie zal dan opgevangen worden in de lagen daarboven.

Een belangrijke regel bij Value objects is dat deze geen identiteit bevatten, ze hebben geen unieke identifier en zijn daarom geen entiteit. Value objects kunnen wel tot een entiteit behoren zoals bijvoorbeeld een User entiteit.

Value objects bevatten ook bepaalde logica of methoden die met dat object gepaard gaan.

Value objects geven ook betekenis aan de code. In het volgende code voorbeeld is het niet meteen duidelijk welke parameter de verzender is, en welke de ontvanger is. Een mogelijke oplossing is goede namen geven aan de parameters.

Listing 5.1: Een minder goed voorbeeld van Value Objects

```
1 // 1. slecht
2 class AddAppointment {
3   private String person1;
4   private String person2;
5   private DateTime date;
6
7   // Wanneer er een slechte naamkeuze wordt gemaakt in de naam
8   // van de variabele, kan dit tot bugs leiden.
9   public AddAppointment(String person1, String person2, DateTime date) {
```

```
10      this.person1 = person1;
11      this.person2 = person2;
12      this.date = date;
13    }
14  }
15
16  // 2. beter
17  class AddAppointment {
18    private String agent;
19    private String client;
20    private DateTime date;
21
22    // Dit is al beter leesbaar, het gevaar bestaat er in agent & client
23    // Van plaats te verwisselen. Bij 2 geldige strings zal dit dus geen
24    // Exceptie gooien
25    public AddAppointment(String agent, String client, DateTime date) {
26      this.agent = agent;
27      this.client = client;
28      this.date = date;
29    }
30  }
```

Wanneer er nu een AddAppointment object gemaakt wordt, is er wel nog steeds het gevaar de agent en de klant om te draaien. Als we geldige strings gebruiken, zal de code geen exceptie gooien en zit er een bug in het systeem.

Dit probleem kan opgelost worden door Value objects te gebruiken. Bij het verwisselen van de parameters zal er een exceptie gegooid worden omdat de types niet overeen komen.

Listing 5.2: Een voorbeeld van Value Objects

```
1   class Id {
2     private GUID id;
3
4     public Id(GUID id) {
5       this.id = id;
6     }
7
8     public GUID getId() {
9       return this.id;
10    }
11
12    public void setId(GUID id) {
13      this.id = id;
14    }
15  }
16
17  class ClientId extends Id {}
18  class AgentId extends Id {}
19
20  class AddAppointment {
21    private AgentId agent;
22    private ClientId client;
23    private DateTime date;
24
25    public AddAppointment(AgentId agent, ClientId client, DateTime date) {
26      this.agent = agent;
27      this.client = client;
28      this.date = date;
29    }
30  }
```

5.3 Messages

Communicatie in een applicatie draait rond messages. Er zijn 3 soorten messages die terug komen in een applicatie: informational, interrogatory en imperative (Verraes, 2015). Binnen een applicatie met EventSourcing wordt er ook gebruik gemaakt van messages.

5.3.1 Imperative Messages

De imperative of imperatieve messages, zijn messages die de intentie hebben om de ontvanger van dit bericht een actie te laten uitvoeren. Binnen EventSourcing is een typisch voorbeeld een Command, waarbij het Command de intentie heeft om een actie te laten uitvoeren door zijn Command Handler. Binnen een systeem zijn deze Commands iets typisch dat uit de businesskant komt. Wanneer het volgende voorbeeld door iemand van de business gelezen wordt, weet die ook meteen wat er zal gebeuren, als er gewerkt wordt met deze Command.

Listing 5.3: Een voorbeeld van een Command

```
1  class ChangeAppointmentStartDate {
2    private AppointmentId appointmentId;
3    private DateTime start;
4
5    public ChangeAppointmentStartDate(
6      AppointmentId appointmentId,
7      DateTime start
8    ) {
9      this.appointmentId = appointmentId;
10     this.start = start;
11   }
12
13   public AppointmentId getAppointmentId() {
14     return this.appointmentId;
15   }
16
17   public DateTime getStart() {
18     return this.start;
19   }
20 }
```

Wanneer er met deze klasse gewerkt wordt (of met objecten van deze klasse), dan wordt er verwacht dat de ontvanger iets zal wijzigen.

5.3.2 Interrogatory Messages

De interrogatory of ondervragende messages, zijn messages die iets vragen over de huidige state van de applicatie. In elke applicatie worden er zaken opgevraagd. Binnen EventSourcing wordt dit ook gedaan aan de hand van Querys, deze Querys gaan informatie opvragen van de huidige state. Deze Querys worden ook opgelegd door de businesskant, de business kan, wanneer ze het voorbeeld zien, meteen begrijpen wat er zal gebeuren als er met deze Query wordt gewerkt omdat het een 1-op-1 vertaling is van de business.

Listing 5.4: Een voorbeeld van een Query

```
class MyAppointmentsForToday {
  private UserId userId;
  private Date today;

  public MyAppointmentsForToday(UserId userId) {
    this.userId = userId;
    this.today = new Date();
  }
}
```

Bij dit voorbeeld is het duidelijk dat er informatie opgevraagd wordt en dat er geen wijzigingen aan de huidige state zullen gebeuren.

5.3.3 Informational Messages

De informational of de informatieve messages, zijn messages die iets over zichzelf willen vertellen. Bij EventSourcing wordt dit uitgedrukt in DomainEvents. DomainEvents zijn de messages die opgeslagen zullen worden in de databank. DomainEvents worden ook meestal in de verleden tijd geschreven omdat ze benadrukken dat er iets gebeurd is.

Listing 5.5: Een voorbeeld van een DomainEvent

```
class AppointmentsStartDateHasBeenChanged {
  private AppointmentId appointmentId;
  private DateTime start;

  public AppointmentsStartDateHasBeenChanged(
    AppointmentId appointmentId,
    DateTime start
  ) {
    this.appointmentId = appointmentId;
    this.start = start;
  }

  public AppointmentId getAppointmentId() {
    return this.appointmentId;
  }

  public DateTime getStart() {
    return this.start;
  }
}
```

Dit voorbeeld lijkt sterk op het voorbeeld van de Command, maar de intentie van beide klassen is totaal verschillend. De klasnamen zijn altijd expliciet, soms zijn ze wat lang maar het is duidelijk wat de bedoeling is. Doordat de messages zo expliciet geschreven zijn, kan er gemakkelijk mee gecommuniceerd worden wat het hele doel is van messages.

Er mag ook geen gemeenschappelijke naam gezocht worden voor deze klassen om 'dubbele code' te verwijderen. Het zijn totaal aparte concepten, zelfs wanneer verschillende Commands tot hetzelfde resultaat leiden, moet dit niet vervangen worden door 1 Command (Verraes, 2014). Vanaf dit gebeurt, komen er problemen naar boven. Wat als er extra informatie in een Command moet maar niet in een DomainEvent of vice versa? De leesbaarheid van de code gaat ook achteruit, de communicatie wordt onduidelijk.

5.4 Domain Events

Domain events zijn een essentieel onderdeel van EventSourcing. Domain Events zijn, net als commands and queries, messages. Domain Events zijn de effectieve events die zullen opgeslagen worden in een append-only database. Een event is iets dat gebeurd is en nooit meer kan veranderen. Er zijn een paar belangrijke eigenschappen van deze events.

- Ze bevatten een unieke id, die op voorhand vastgelegd is. Dit kan een Globally Unique Identifier (GUID) zijn.
- Ze bevatten enkel de data die gewijzigd is ten opzichte van de vorige versie.
- Alle data die ze bevatten, is correct en kan niet meer aangepast worden.

Domain events worden opgeslagen in een append-only database, maar wat als er een fout gemaakt is? Indien een fout is opgetreden, moet er een nieuw domain event gemaakt worden, dat het vorige event corrigeert. Op deze manier blijft al de data correct, en gaat er geen belangrijke informatie verloren. Er is ook niet geprutst met de historiek van deze events, wat van belang is omdat de geschiedenis herschrijven gevolgen kan hebben voor een systeem. Het herschrijven van de geschiedenis houdt in dat de audit log (Hoofdstuk 5.8) niet meer geldig is. Het houdt ook in dat elke projectie opnieuw zal moeten opgebouwd, omdat een wijziging in 1 event een totaal andere uitkomst kan creëren in de huidige state.

Domain events hebben ook een naam, deze naam is specifiek voor de use case. Het vertelt meer over wat er gebeurd is of wat er zal gebeuren. Het is ook aangeraden dat deze naam in de verleden tijd is opgesteld. Zo kan er gedifferentieerd worden tussen de soorten messages (Hoofdstuk 5.3) en kan de leesbaarheid verhoogd worden naar andere partijen toe (developers, business). Een event is tenslotte gebeurd in het verleden. Als er in context van Skedify gesproken wordt, dan is AppointmentWasRescheduled een goede naam voor een domain event. Het is ook van belang dat er geen Create Read Update Delete (CRUD) events gemaakt worden zoals “OfficeWasCreated”, want er werd niet effectief een office gemaakt in het echte leven. Er is echter een office, die in het echt bestaat, aan de applicatie toegevoegd. “OfficeWasAdded” zou dus een betere naam zijn.

5.5 Invariants

Invariants zijn regels die opgelegd zijn door de business. Invariants worden gecontroleerd alvorens een domain event opgeslagen wordt. Elke invariant moet goedgekeurd worden, want eens een domain event opgeslagen is, kan dit niet meer ongedaan gemaakt worden. Er kan wel een nieuw domain event opgeslagen worden om deze wijziging teniet te doen. Invariant controle wordt uitgevoerd bij het triggeren van een command, telkens wanneer een input niet aan deze invariant voldoet moet er een exception gegooid worden. Op deze manier kunnen er inconsistenties opgevangen worden.

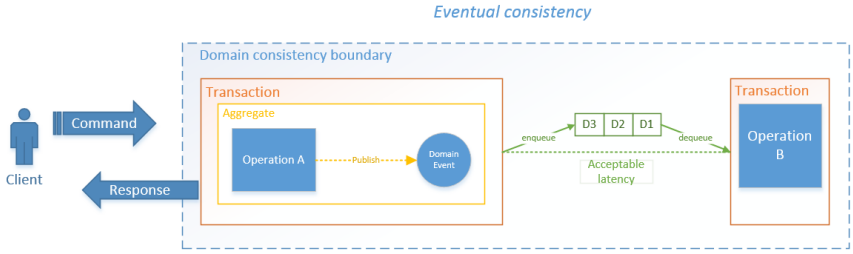
Invariants kunnen ook gecontroleerd worden in Value Objects (hoofdstuk 5.2) en ook deze zullen een exception gooien indien er een probleem optreedt. Een exceptie is een zeer eenvoudige manier om de applicatie vroegtijdig te laten stoppen en een bericht aan de

gebruiker terug te geven.

5.6 Eventual Consistency

Elk commando dat in het systeem terecht komt, kan op een queue geplaatst worden. Het grote voordeel hiervan is dat het systeem niet overbelast wordt en dat elke queue job afgehandeld kan worden (King, 2015). Vandaar de term eventual consistency, uiteindelijk zal het systeem consistent zijn. Een ander groot voordeel is dat eventual consistency automatisch samen kan werken met schaalbaarheid van een applicatie, zoals beschreven in Hoofdstuk 6. Wanneer er een commando uitgevoerd wordt op het systeem, kent de client reeds de nodige data die hij net verstuurd heeft. Er is momenteel geen nood om meteen een respons terug te krijgen. Daarom is het ook belangrijk bij commands om een GUID op voorhand te definiëren zodat de identiteit al bekend is. Met deze identiteit kan er eventueel al een pagina getoond worden waar de nieuwe data weergegeven wordt.

Figuur 5.1: Eventual Consistency



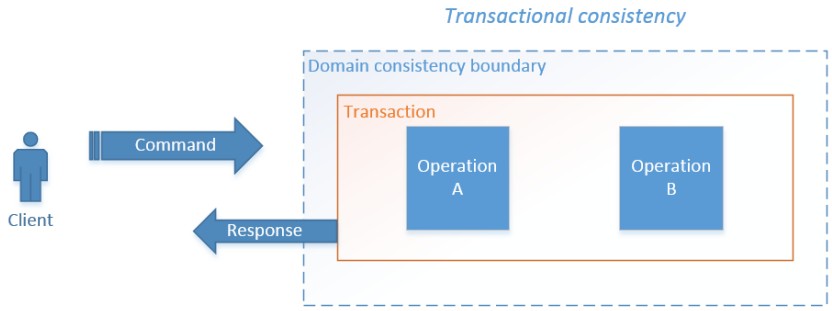
5.6.1 Transactional Consistency

Het voordeel aan transactional consistency is dat de respons meteen terug gegeven wordt. Alle delen van de applicatie zijn doorlopen en alle gegevens zijn gepersisteerd. De grote nadelen zijn dat de applicatie volledig doorlopen moet worden of dat de applicatie een exponentieel hoog aantal requests binnen krijgt en dus alles meteen moet afhandelen. Hierdoor kan de respons veel trager zijn en dit kan resulteren in het crashen van de applicatie. Transactional consistency is ook moeilijker schaalbaar omdat alles in 1 keer uitgevoerd wordt.

5.7 Event Store

De EventStore is een relevant concept bij EventSourcing. De EventStore is de plaats waar alle Domain Events opgeslagen zullen worden. De EventStore is een append-only databank waar enkel nieuwe events aan toegevoegd kunnen worden, maar geen events kunnen verwijderd worden. Om een Domain Event ongedaan te maken, moet er een nieuw event opgeslagen worden die het oude event teniet doet. De EventStore kan elk soort databank zijn, er is ook een speciale databank ontwikkeld die is te vinden op

Figuur 5.2: Transactional Consistency



https://geteventstore.com. Voor deze bachelorproef, en de proof-of-concept zal er gebruik gemaakt worden van een simpele EventStore in MySQL. Een EventStore is helemaal niet zo moeilijk qua structuur, het bevat volgende minimale velden:

- id, een simpele id, die autoincrementeel kan zijn
- stream_id, een id die bij een Aggregate hoort, dit is een GUID. Alle events die behoren tot een bepaalde aggregate zullen dezelfde stream_id hebben.
- stream_version, een versie die incrementeel is per aggregate. Dit zorgt ervoor dat events in de juiste volgorde kunnen worden afgespeeld indien nodig.
- event_name, de naam van het DomainEvent dat opgeslagen moet worden. Via deze naam kan het juiste object weer opgebouwd worden in de code.
- payload, dit bevat de effectieve data van het DomainEvent. Dit kan een JavaScript Object Notation (JSON) object zijn. Naast de naam wordt dit mee gebruikt om het object weer op te bouwen.
- recorded_at, een timestamp waarop het event opgeslagen is geweest. Dit kan handig zijn voor de audit log.

5.8 Audit Log

Wanneer er gebruik gemaakt wordt van EventSourcing, is er automatisch een gratis audit log beschikbaar omdat de EventStore append-only is. Deze audit log is een bewijs van alle events die tot een bepaalde waarde (projection) leiden. Wanneer de EventStore op een Write Once Read Many (WORM) drive gezet wordt, dan kan er niet geknoeid worden met deze lijst van events. Het voordeel hieraan is dat er gemakkelijk bewijs kan geleverd worden naar andere partijen indien dat nodig is. Stel dat er een rechtzaak komt tussen 2 bedrijven omdat er geknoeid is met de data, dan kan er zwart op wit bewezen worden dat er niet geknoeid is met de data.

Deze audit-log heeft ook nog andere voordelen, als er een fout zit in de applicatie en de applicatie moet gedebugged worden, dan kan men de audit log nemen en deze opnieuw afspelen tot de huidige staat. Zo kan er per event gekeken worden of de uitkomst van de huidige staat al dan niet correct is.

5.9 Projections

Projections zijn een projectie van de huidige staat die berekend is uit alle voorgaande events. Bij een systeem waar er met een relationele databank wordt gewerkt, is de databank zelf de projectie, met als grote verschil dat daar ook de databank de enige echte “single source of truth” is. Dat is niet het geval bij een systeem met EventSourcing, daar is de “single source of truth” de events die opgeslagen zijn.

Projections hebben het voordeel dat er meerdere kunnen zijn. Er kan een databank gegenereerd worden voor de businesskant die dan allerlei statistieken kan bekijken. Er kan ook een databank gegenereerd worden voor werknemers die met een bepaalde applicatie werken. Er kan zelfs een databank of ander opslagsysteem gevuld worden, los van de huidige databanken voor de andere partijen. Op deze manier zijn het systeem en de databank losgekoppeld van elkaar en dit door de flexibele projecties.

Deze projecties zijn een berekening uit de voorbije events, dit wilt ook zeggen dat er niet geoptimaliseerd moet worden voor de schrijfkant (Hoofdstuk 4). Er kan wel geoptimaliseerd worden voor de leeskant. Er is geen nood om een aantal tabellen elke keer met elkaar te gaan joinen[1], als er rechtstreeks naar 1 tabel geschreven wordt.

Om een wijziging te kunnen doorvoeren in een aggregate, moet eerst de huidige state opgebouwd worden vanuit de DomainEvents. Dit kan echter inefficiënt zijn wanneer er duizenden events beschikbaar zijn. Een projectie kan dan perfect dienen als een Cache, om dit probleem op te lossen.

5.10 Testen

Zoals besproken in Hoofdstuk 5.3, zijn alle queries en commando’s klassen die leesbaar zijn, en de naamgeving komt rechtstreeks van de businesskant. Testen schrijven voor de domain logica zoals commando’s en queries kunnen ook opgesteld worden door de business en gevalideerd worden door hen. De volgende structuur van testen kan gebruikt worden:

- Given: een lijst van events
- When: een commando uitgevoerd wordt
- Then: worden er Domain Events (Hoofdstuk 5.4) geretourneerd of excepties gegooid (Hoofdstuk 5.5)

Deze manier van testen levert automatisch documentatie op die gevalideerd kan worden door de business. Wanneer er nieuwe developers bijkomen, kunnen ze door het raadplegen van de automatische testen, te weten komen hoe bepaalde businessregels werken.

[1] Het samenvoegen van twee tabellen via de “JOIN” instructie binnen de SQL taal.

5.10.1 Debugging

Het debuggen van een EventSourced systeem kan op volgende manier gebeuren: alle events worden opgeslagen, een developer kan dan een reeks aan events programmeren of kopiëren uit een Log. De developer heeft dan de mogelijkheid om stap voor stap het proces te doorlopen met de data uit de Log, om zo te achterhalen waar het probleem zich bevond.

6. Schaalbaarheid

Een systeem dat gebruik maakt van EventSourcing, kan geschaald worden. Het schalen is relatief gemakkelijk door volgende redenen. De EventStore is een append-only database, er kan altijd een kopie genomen worden van deze database. Er moet niet gekeken worden of er al dan niet wijzigingen zijn in de data en welke de correcte data is. Synchroniseren van meerdere EventStores is een kwestie van ontbrekende rijen aan te vullen. Hiervoor kan er een mechanisme gebruikt worden zoals "geef alle events na id X[1]". Schalen van de applicatie zelf kan als volgt gebeuren: er kunnen projecties bijgemaakt worden die zichzelf gaan 'voeden' met alle events uit de EventStore. Van zodra ze up-to-date zijn met de EventStore kunnen ze als projection (zie Hoofdstuk 5.9) geregistreerd worden, om live naar de events te luisteren.

[1] een onbekende variabele

7. Voordelen

Er zijn meerdere voordelen verbonden aan EventSourcing. Een voordeel is dat er van technologie gewisseld kan worden voor de projecties. Stel dat er gebruik gemaakt werd van MySQL om de huidige staat in op te slaan, dan kan, wanneer er gemerkt wordt dat er nood is aan een Graph database database in plaats van een Relational Database Management System, er een projectie bijgemaakt worden die simultaan met de MySQL database loopt. Zodra de Graph database database volledig gevoed is met de events uit de EventStore, dan kan de MySQL databank weggegooid worden. Doordat een projectie opgebouwd is in code, en in een versiebeheersysteem zit, moet er niet nagedacht worden over het nemen van een eventuele back-up van deze MySQL databank omdat ze opnieuw opgebouwd kan worden door middel van voorgaande events. Indien deze databank later opnieuw ter beschikking moet staan, dan kan deze projectie terug ingeladen worden en volledig opnieuw opgebouwd worden. Het is mogelijk om een Graph database database in te voeren in de applicatie die bijvoorbeeld al meer dan 3 jaar actief is. Zodra de events volledig gevoed zijn door middel van een projectie in de Graph database database, dan staat deze op het punt alsof ze al van in het begin van de applicatie geprogrammeerd was. Dit is het grote voordeel van al de events bij te houden. Een ander voordeel van EventSourcing is dat enkel de EventStore gebackupped moet worden. Indien elke server corrupt of kapot gaat en alle MySQL instanties kapot zijn, dan kunnen deze opnieuw opgebouwd worden. Dit is mogelijk omdat de EventStore de single source of truth is.

8. Nadelen

...TODO...

9. Kosten

De kosten van EventSourcing kunnen opgedeeld worden in 3 grote delen. Het eerste deel gaat over de fysieke infrastructuur kosten die EventSourcing met zich meebrengt. Het opslaan van alle belangrijke businessdata doorheen de jaren dat de applicatie werkt, brengt bepaalde kosten met zich mee. Het tweede deel gaat over de kosten van de developers, EventSourcing is een andere manier om met gegevens om te gaan en dus ook een andere manier om code te schrijven. Tot slot gaat het derde deel over de kost van performance aangezien elk event overlopen moet worden, kan dit een performance probleem met zich mee brengen.

9.0.1 Infrastructurele kosten

Er van uitgaande dat events opgeslagen worden als JSON objecten kan hier vrij snel een berekening op gedaan worden. Hier zijn volgende zaken waar er rekening mee gehouden moet worden: grootte van het JSON object en het aantal requests. De grootte van een JSON object voor een bepaald event is niet zo groot. Om dit te weten te komen, wordt de grootte van de Payload om een nieuwe afspraak te maken genomen en als JSON object opgeslagen. Nu we dit event hebben opgeslagen, kan de grootte in bytes uitgelezen worden. De keuze om de payload van een nieuwe appointment te gebruiken is omdat deze het meest voorkomt in de applicatie van Skedify. De grootte van dit event bedraagt 168 bytes. Uit informatie van New Relic, een platform dat door Skedify gebruikt wordt om aan monitoring te doen, blijkt dat er in 1 maand tijd 143500 requests worden uitgevoerd die een schrijfactie uitvoeren. Met deze gegevens kan er berekend worden hoeveel disk capaciteit

er nodig is om deze events voor lange tijd bij te houden.

$$\frac{143500 \text{ requests}}{\text{maand}} \times 168 \text{ bytes} \times 5 \text{ jaar} = 1.37 \text{ GB} \tag{9.1}$$

Dit wilt zeggen dat er 1.37GB plaats moet voorzien worden als men de events wilt opslaan voor 5 jaar. In de volgende tabel zal er een assumptie gemaakt worden dat het aantal requests doorheen de jaren zal stijgen en dat het aantal bytes voor een event ook zal stijgen.

Tabel 9.1: Disk capaciteit nodig om alle events op te slaan[1]

Requests / maand	Event grootte	Periode	Nodige disk capaciteit
150 000	500 bytes	10 jaar	8.5 GB
200 000	750 bytes	10 jaar	17.01 GB
300 000	1 kB	10 jaar	34.83 GB
350 000	1.5 kB	10 jaar	60.69 GB
400 000	1.5 kB	10 jaar	69.67 GB
500 000	1.5 kB	10 jaar	87.08 GB
800 000	2 kB	50 jaar	928.88 GB

Zoals te zien in tabel 9.1 is het helemaal geen probleem om de events op te slaan omdat deze helemaal niet zo groot zijn. De kost is 1 harde schijf extra, die elk jaar goedkoper en goedkoper wordt. Meer nog, zolang de datahoeveelheid onder Kryder's law blijft, wordt het data opslagmechanisme alleen maar goedkoper over tijd.

Een tweede kost waar er rekening mee moet worden gehouden zijn backups. Zoals vermeld in Hoofdstuk 5.9 zijn de databanken die gebruikt worden projecties van de afgebeelde events. Vermits deze een berekening zijn, volstaat het om enkel van de events een backup te nemen, omdat de projecties opnieuw berekend kunnen worden[2].

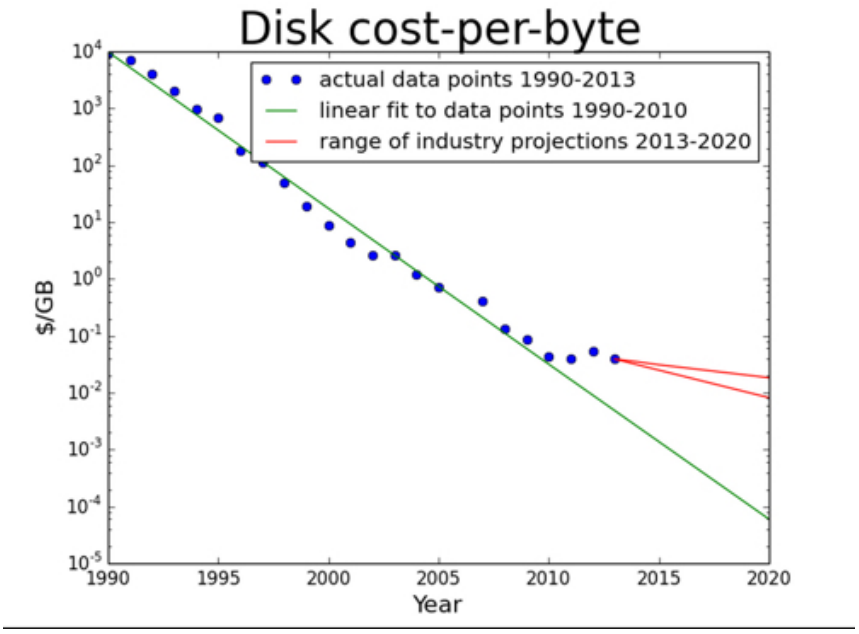
9.0.2 Developer kosten

Het is een andere manier van werken dus zullen de developers ook op de hoogte gebracht worden hoe ze met deze technologie moeten omgaan. Er zijn verschillende resources beschikbaar, waaronder https://eventsourcery.com. Zowel bij Skedify, als bij andere bedrijven wordt er gebruik gemaakt van Object Oriented Programming (OOP). EventSourcing gaat erg goed samen met functional programming omdat de huidige state een Left Fold is

[1] De gegevens staan voor de herkenbaarheid met het kilobyte en gigabyte symbool, maar eigenlijk wordt er kibibyte en gibibyte bedoelt.

[2] Er mag niet vergeten worden om een backup te nemen van data die niet in de databank wordt opgeslagen, zoals afbeeldingen

Figuur 9.1: Kryder slowdown. Chart by Preeti Gupta at UCSC.



van vorige events.

$$\text{current state} = f(\text{events}, \text{action}) \tag{9.2}$$

Functioneel leren programmeren is dus een meerwaarde.

9.0.3 Performance kosten

Telkens wanneer er een nieuwe command uitgevoerd wordt zoals vernoemd in Hoofdstuk 5.3.1, moeten alle events uit de EventStore overlopen worden om de huidige staat van het object op te bouwen. Dit wil zeggen dat bij elk nieuw event, de applicatie trager zal worden. In kleine en middelmatige applicaties heeft dit geen gevolgen, naar grotere applicaties toe heeft dit grotere gevolgen. Hiervoor is een oplossing van snapshots. Een snapshot is de huidige state van een object na het overlopen van x aantal events. Deze kunnen in de EventStore opgeslagen worden, in een tabel naast de events zelf. Telkens wanneer de klasse beschrijving zou veranderen, moeten de snapshots opnieuw gegenereerd worden. Een tweede optie is om de current state ook op te slaan in memory, via deze weg moeten er geen events opnieuw afgespeeld worden om de huidige state van een object te kunnen bepalen omdat deze reeds beschikbaar is.

10. Proof of concept

...TODO...

11. Conclusie

...TODO...

Lijst van acroniemen

CQRS Command Query Responsibility Segregation. 3, 5, 13, 14, 17, 19
CQS Command Query Separation. 3, 17, 18
CRUD Create Read Update Delete. 26

DDD Domain Driven Design/Development. 5, 22

GUID Globally Unique Identifier. 26–28

JSON JavaScript Object Notation. 28, 37

OOP Object Oriented Programming. 38
ORM Object Relational Mapper. 19

RDBMS Relational Database Management System. 3, 11, 33, 47

SQL Structured Query Language. 19, 29

WORM Write Once Read Many. 28

Verklarende Woordenlijst

Cache Caching is het principe van het resultaat van berekingen op te slaan, zodat de bereking niet opnieuw gedaan moet worden. 29

Caching Caching is het principe van het resultaat van berekingen op te slaan, zodat de bereking niet opnieuw gedaan moet worden. 21

Command Een message die de intentie heeft een actie te laten uitvoeren, “doe dit”, “doe dat”. 24, 25

Command Handler Een object die de taak van beschreven in een command, zal uitvoeren. 24

Getter Een getter is een methode die informatie uit een object ophaalt, zonder dit object te muteren. 17, 18

Graph database Een database waarbij er grafen structuren worden gebruikt om gegevens op te slaan in plaats van rijen en kolommen, ze maken gebruik van nodes en edges. 33

Left Fold Een left fold is een functie die een lijst door middel van een andere functie reduceert tot 1 uitkomst. 38

Log Een log file is een verzameling of een opname van evenementen die gebeurd zijn in een systeem. 3, 30

Meetup Mini conferenties waar 2 à 3-tal mensen een presentatie gegeven omtrent een bepaald onderwerp. 5

MySQL MySQL is een managementsysteem voor relationele databases (RDBMS), SQL is de taal die wordt gebruikt om een database van dit systeem op te bouwen, te bevragen en te onderhouden. 28, 33

Payload Een geheel aan informatie bedoeld getransporteert te worden. 37

Query Een message die de intentie heeft informatie op te vragen uit het systeem. 24

Setter Een setter is een methode die geen informatie uit een object ophaalt, maar het object muteert. 17, 18

Void Een type dat “niets” betekent. 17

12. Bijlage A: Bachelorproefvoorstel

EventSourcing, een manier om wijzigingen in data op te slaan als events, bij Webapplicaties.

Onderzoeksvoorstel Bachelorproef

Robin Malfait[1], Mathias Verraes [2]

Samenvatting
Bedrijven leven van data, geen enkel bedrijf kan in de toekomst kijken. Daarom is het belangrijk om al de data over de loop der jaren bij te houden om de meest accurate rapporten te genereren. EventSourcing kan hierbij helpen, het is een manier om data op te slaan, onder de vorm van events. De huidige staat van de applicatie kan opgebouwd worden aan de hand van deze events. Terug in de tijd gaan is met EventSourcing geen enkel probleem. EventSourcing zou meer gebruikt moeten worden omdat het een luxe biedt aan de business om interessante rapporten te genereren. Er zal een onderzoek gebeuren naar het verschil tussen huidige systemen en EventSourcing systemen en of het een meerwaarde biedt naar de business toe. In deze bachelorproef zal de business afgebakend worden tot bedrijven die bezig zijn met online afspraak planning. In dit document kan er al wat meer informatie over EventSourcing gevonden worden, alsook studies van belangrijke personen en organisaties zoals Greg Young en Microsoft. Naar de toekomst toe kan dit onderzoek gebruikt worden om meer informatie te verkrijgen rond EventSourcing en wat hier de voor- en/of nadelen van zijn.

Sleutelwoorden
Webapplicatieontwikkeling — Databeheer — EventSourcing — Events

Contact: [1] robin.malfait.v3534@student.hogent.be; [2] mathias@verraes.net

Inhoudsopgave

1 **Introductie** 1
2 **Onderzoeksvraag** 1
3 **State-of-the-art** 2
4 **Methodologie** 2
5 **Verwachte resultaten** 2
6 **Verwachte conclusies** 2
Referenties 2

1. Introductie

EventSourcing is een manier om wijzigingen in data op te slaan. Om deze wijzigingen te modeleren wordt er gebruik gemaakt van events. Deze events liggen vast en kunnen niet meer gewijzigd worden.

De events worden dan gebruikt om bijvoorbeeld een database op te vullen met de huidige staat. Achteraf kunnen er rapporten gegenereerd worden op basis van deze events. Het voordeel is dat er geen wijzigingen gebeuren, maar dat er altijd events toegevoegd worden aan een lijst of een event store. Op deze manier zal en kan er geen data verloren gaan.

Het is ook belangrijk om te weten dat deze events de 'Single Source of Truth' zijn. Dit wilt zeggen dat elke state afgeleid wordt vanaf deze events.

EventSourcing is een interessante manier om data op te slaan. Zoals eerder vermeld gaat er geen informatie verloren. Dit is zeer interessant voor de business, want niemand kan in de toekomst kijken. De business zal hier concreet gedefinieerd worden als een bedrijf dat gespecialiseerd is in online scheduling, zoals Skedify. Als er in een huidig systeem een update van een cell in een tabel gebeurt, dan gaat de vorige waarde verloren. Niemand zal zich afvragen wat daar ooit stond, tot iemand er om vraagt. Vanaf dit moment is er vraag naar data die er ooit was maar nu niet meer. Dit probleem valt op te lossen door middel van Event Sourcing. Rapporten kunnen vandaag geschreven worden, en bekeken worden alsof het rapport al bestond van vòòr het geschreven werd, dit soort tijdreizen is dan weer mogelijk omdat er geen data verloren gaat.

2. Onderzoeksvraag

De onderzoeksvraag luidt als volgt: Wanneer is EventSourcing een meerwaarde voor een bedrijf zoals Skedify, waar ze gespecialiseerd zijn in online scheduling?

Met als sub vragen:

- Wat is het verschil tussen een huidig systeem en een systeem met EventSourcing?
- Hoe kan een applicatie die gebruik maakt van EventSourcing getest worden?
- Welke delen moeten er EventSourced worden, en hoe kan je deze delen bepalen?

3. State-of-the-art

EventSourcing is een vrij recent idee binnen de informatica. Het concept op zich bestaat al veel langer, denk aan dokters die enkel informatie toevoegen aan een dossier van een patient, of aan de wet waar enkel documenten aan toegevoegd worden met nieuwe wetten of verbeteringen op wetten. Greg Young is een van de grondleggers van EventSourcing. Hij heeft zelf al veel onderzoek gedaan naar EventSourcing, hij vertelt hier meer over in een conferentie presentatie Young, 2014. Daarnaast is er ook een EventStore die heel populair is binnen de EventSourcing wereld, geteventstore.com. Het is een databank speciaal gebouwd en geoptimaliseerd voor systemen met EventSourcing. Naast Greg Young is er een bekender bedrijf, Microsoft, die EventSourcing als een pattern beschrijft. Dit is uitgeschreven in een blog Microsoft, 2015 en hier wordt veel naar verwezen als er over EventSourcing gesproken wordt. Tot slot is er Martin Fowler, Martin is een topman binnen de informatica die veel onderzoek verricht. Hij is beter bekend van het agile manifesto en tal van andere zaken. Hij heeft een onderzoek gedaan naar de werking van EventSourcing wat terug te vinden valt op zijn blog Fowler, 2005.

4. Methodologie

Heeft een software bedrijf effectief deze technieken nodig? Dit zal worden onderzocht aan de hand van een onderzoek bij Skedify. Skedify is een bedrijf dat open staat voor dergelijke onderzoeken. Om dit in goede banen te leiden, zal er een veldonderzoek uitgevoerd worden bij de business van Skedify. Dit zal dan bepalen wat de vereisten zijn en wat de mogelijke nadelen zijn van een systeem met EventSourcing. Het veldonderzoek zal dan een antwoord kunnen geven op de onderzoeksvraag: wanneer is EventSourcing een meerwaarde voor een bedrijf zoals Skedify waar ze gespecialiseerd zijn in online scheduling? Naast het veldonderzoek zal er ook een studie gebeuren naar de kosten van opslag van de data en van de performantie van een systeem met EventSourcing.

Er zal een proof-of-concept ontwikkeld worden in de programmeertaal php. Deze toepassing zal een webapplicatie zijn die de basis ideeën van Skedify gebruikt met behulp van EventSourcing. Deze proof-of-concept zou volgende zaken duidelijk moeten maken:

- Wat is het verschil tussen een huidig systeem en een systeem met EventSourcing?
- Hoe kan een applicatie die gebruik maakt van EventSourcing getest worden?

5. Verwachte resultaten

Er wordt verwacht aan de hand van het veldonderzoek dat er een interesse gaat zijn naar systemen met EventSourcing. Er zal waarschijnlijk wel een nadeel naar boven komen dat het een andere manier van werken is waardoor resources moeilijker te vinden zijn. Developers zullen ook opgeleid moeten worden om met dit soort systemen te werken, dit kost geld en zal dus een doorslag kunnen geven in de resultaten of EventSourcing al dan niet een meerwaarde biedt.

Bij het onderzoek naar de performantie van een EventSourcing systeem, en naar de kosten van data opslag wordt er verwacht dat deze geen invloed zullen hebben om te bepalen of EventSourcing een meerwaarde biedt. Data opslag is tegenwoordig zeer goedkoop en zal alleen maar goedkoper worden.

6. Verwachte conclusies

- Wat is het verschil tussen een huidig systeem en een systeem met EventSourcing?
  Hierbij wordt er verwacht dat een systeem met EventSourcing wat complexer gaat zijn, maar veel interessanter naar de business toe omdat er geen data verloren gaat. De manier van code schrijven zal ook verschillend zijn omdat het fundamenteel anders is.
- Wanneer is EventSourcing een meerwaarde voor een bedrijf zoals Skedify waar ze gespecialiseerd zijn in online scheduling?
  Er is een fundamenteel verschil tussen green field (start ups) en brown field (oudere) bedrijven. Bij een start up is er nog een ruime keuze aan technologieën, bij een brown field bedrijf wordt integratie of overgang naar een ander system veel moeilijker. Skedify is een start up, waardoor de drempel van vastgelegde technologieën niet in de weg staat voor nieuwe technologieën zoals EventSourcing. Daarnaast zal het ook een meerwaarde bieden wanneer de kosten voor een opleiding van de developers om met EventSourcing te werken kleiner is dan de kost van verloren data. Door een volledige historiek te kunnen bekijken wordt de toekomst voorspelbaarder en zal dit een meerwaarde bieden voor het bedrijf.
- Hoe kan een applicatie die gebruik maakt van EventSourcing getest worden?
  Er is geen huidige staat die getest kan worden dus er zal getest moeten worden of er effectief events opgeslaan worden en dat deze juist zijn.
- Welke delen moeten er EventSourced worden, en hoe kan je deze delen bepalen?
  Waarschijnlijk moeten vooral de delen met business waarde EventSourced worden. Een druk op een knop of een login van een gebruiker zullen waarschijnlijk niet EventSourced moeten worden. Het aanvragen van een afspraak zal dan weer wel EventSourced moeten worden.

Referenties

Fowler, M. (2005, december). Event sourcing - capture all changes to an application state as a sequence of events. Laatst bezocht op 06/12/2016. Verkregen van http://martinfowler.com/eaaDev/EventSourcing.html

Microsoft. (2015). Event sourcing pattern. Laatst bezocht op 06/12/2016. Verkregen van https://msdn.microsoft.com/en-us/library/dn589792.aspx

Young, G. (2014, augustus). Greg young - cqrs and event sourcing - code on the beach 2014. Laatst bezocht op 06/12/2016. Verkregen van https://www.youtube.com/watch?v=JHGkaShoyNs

Bibliografie

Fowler, M. (2005a, december). Command query separation. https://www.martinfowler.com/bliki/CommandQuerySeparation.html. Blog. Last visited on 2017-04-20.

Fowler, M. (2005b, december). Ddd aggregate. https://www.martinfowler.com/bliki/DDD_Aggregate.html. Blog. Last visited on 2017-04-20.

King, J. (2015, juli). Jef king - eventual consistency. https://www.youtube.com/watch?v=fIfH-kUaX4c. YouTube. Last visited on 2017-05-13.

Meyer, B. (1988). *Object-oriented software construction*. United States: Prentice Hall.

Microsoft. (2017, maart). Event sourcing. https://docs.microsoft.com/en-us/azure/architecture/patterns/event-sourcing. Blog. Last visited on 2017-04-20.

Loi - Wet. (g.d.). http://www.ejustice.just.fgov.be/wet/wet.htm. Last visited on 2017-04-20.

Verraes, M. (2014, januari). Domain-driven design is linguistic - just because two behaviours lead to the same outcome, does not mean they are replaceable. http://verraes.net/2014/01/domain-driven-design-is-linguistic/. Blog. Last visited on 2017-03-22.

Verraes, M. (2015, januari). Messaging flavours - differentiating between informational, interrogatory, and imperative messages, and keeping them nicely separated. http://verraes.net/2015/01/messaging-flavours/. Blog. Last visited on 2017-03-19.

Young, G. (2014, september). Greg young - cqrs and event sourcing. https://www.youtube.com/watch?v=JHGkaShoyNs. YouTube. Last visited on 2017-04-19.

Lijst van figuren

4.1 Visuele representatie van tabel 4.1 20

5.1 Eventual Consistency 27
5.2 Transactional Consistency 28

9.1 Kryder slowdown. Chart by Preeti Gupta at UCSC. 39

Lijst van tabellen

4.1 Aantal requests vergeleken voor de lees- en schrijfkant. 19

9.1 Disk capaciteit nodig om alle events op te slaan 38