Who & what
Event Sourcing intro
Event Sourcing architecture
How did we get here?
How do we use it
Experiences
Wrapping up

Event Sourcing at Studyflow.nl

Joost Diepenmaat

February 11, 2015

Who am I

- Joost Diepenmaat
- Last 5 years have been focused on Clojure
- Technical lead @ studyflow.nl
- @ZeekatSoftware

Who & what

Event Sourcing intro

Event Sourcing architecture

How did we get here?

How do we use it

Experiences

Wrapping up

Studyflow

- http://www.studyflow.nl
- Secondary education platform
- Currently providing math courses for over 100 schools

What's in the talk

- What is Event Sourcing
- How we implemented it with Rill
- Our experiences
- Rill implementation details are secondary
    - Maybe for another talk
    - Source code for core library is public

Event sourcing as a concept

- CQRS: writes have different needs from reads (queries)
- Domain Events as system of record / source of truth
- Event Store: append only event streams, read in chronological order

Event Sourcing at Studyflow.nl

Joost Diepenmaat

Who & what

Event Sourcing intro

Event Sourcing architecture

How did we get here?

How do we use it

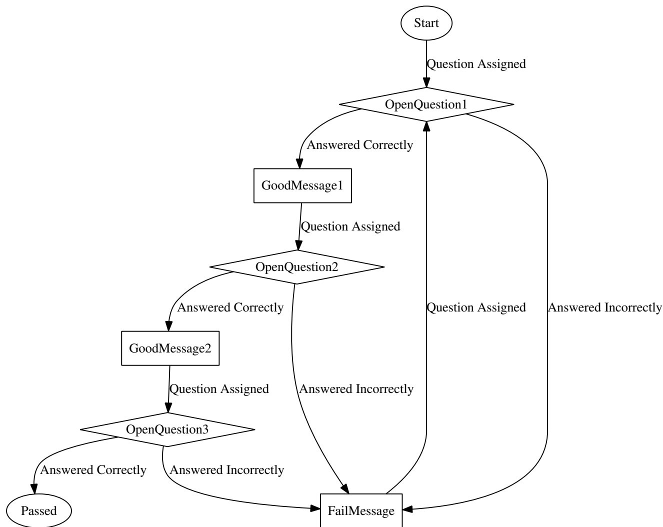
Experiences

Wrapping up

Example: Quiz

- You get assigned a random question
- If you answer correctly, you get shown a message and you can go to the next question
- If you provide a wrong answer, get shown a message and you have to start over
- If you answer 3 questions in a row correctly, you pass

Quiz flow chart



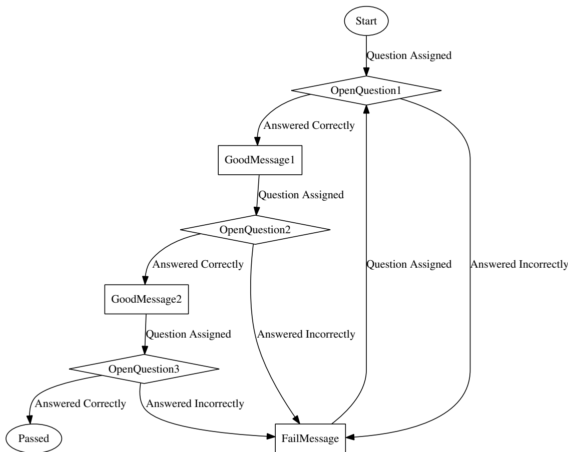
The CRUD strawman

```
{:user-id 1
 :question-id 23
 :answer-state :correct ; or nil or :incorrect
 :question-number 2}
```

What did we lose there?

- Timing: when did user 1 try to answer questions?
- Which questions did user 1 answer?
- How many times did user 1 make a mistake?
- Which mistakes?
- Which questions in our database are difficult?
- What kind of mistakes do users make answering question 45?
- Maybe more stuff *that we will think up in a few months*

Back to the flow chart



What if we store all of these transitions?

Example Domain Event

```
{:rill.message/type
 :quiz/QuestionAnsweredCorrectly,
 :rill.message/timestamp
 #inst "2015-02-11T11:46:55.014-00:00",
 :rill.message/id
 #uuid "276de24f-d7df-478c-a82a-fd97c24a7232",
 :answer "My Answer",
 :question-id 442,
 :user-id 23}
```

Domain Events

- From Domain Driven Design (DDD)
- Record of a thing that *did happen*
- Has *meaning* in the domain
- Records *intent* of change
- Does not change or disappear
- Past tense

Who & what

Event Sourcing intro

Event Sourcing architecture

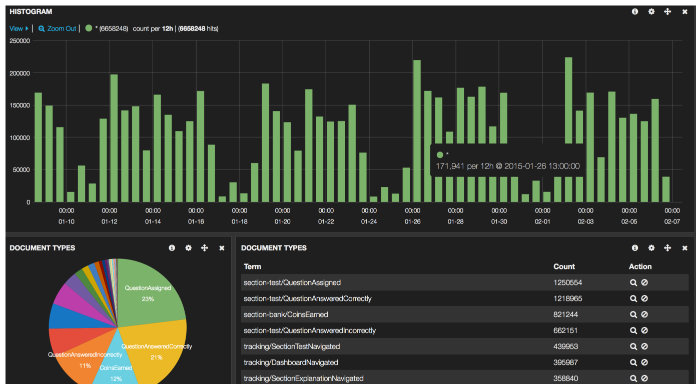
How did we get here?

How do we use it

Experiences

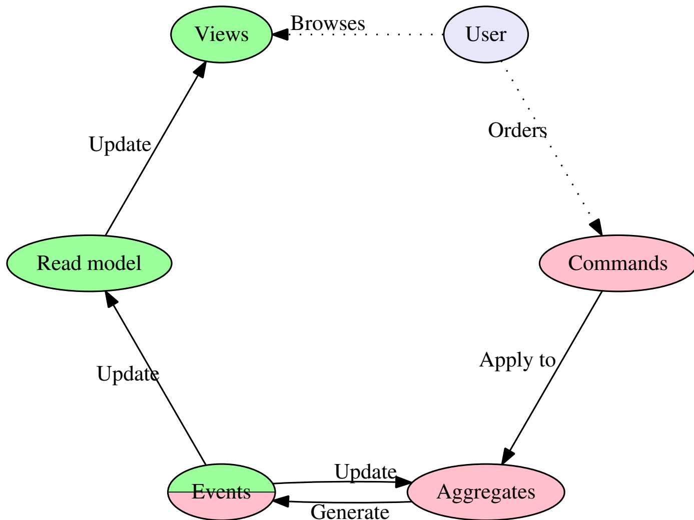
Wrapping up

Our service



- Used by about 25000 students
- Answering 1.2 million questions a month
- Over 5 million domain events per month

Outline



Who & what

Event Sourcing intro

Event Sourcing architecture

How did we get here?

How do we use it

Experiences

Wrapping up

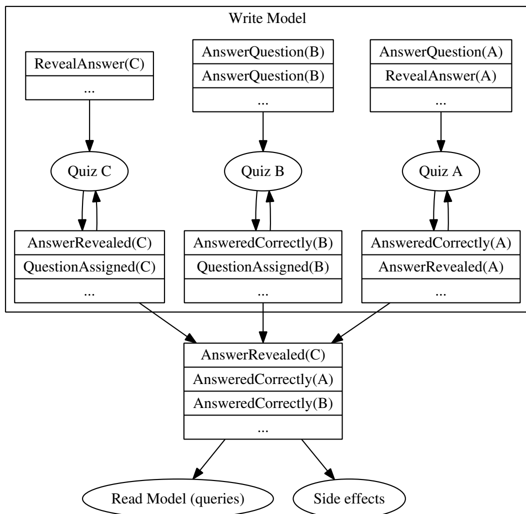
Event sourcing mechanics

- The *write side* state is composed of aggregates
- Commands apply to aggregates to generate events
- Events apply to aggregates to update state
- The *read model* is generated/updated asynchronously from published events

Commands & Events

- Commands either succeed and generate events, or they fail
- Events cannnot fail since you cannot change the past
- Commands are requests, so imperative: "AnswerQuestion!"
- Events are in the past tense: "QuestionAnsweredCorrectly"

Data flow



Why do we use Event Sourcing?

ES architecture matches our core use cases

- track student actions & intents; quite a lot of writes
- potentially complicated reporting
- naturally maps to "real time" event reporting

Advantages

- Simple concepts
- Easily scalable
- One way flow of information
- Queueing, Retrying, Conflict handling..

Who & what

Event Sourcing intro

Event Sourcing architecture

How did we get here?

How do we use it

Experiences

Wrapping up

Core ideas match Clojure's / FP pretty well

- Immutability
- Aggregates as reductions
- Events & Commands are data
- Append-only data store

Technologies we currently use

Clojure web app (ring, hiccup etc)

- Authentication system
- Administration & Teacher front end

Om / Reagent

- Student applications

Ruby on Rails

- Publication & content editing service

Hooked together using event streams

Rill Event Sourcing

- Clojure Event Sourcing toolkit/library
- Developed in house
- EPL license
- Using Postgres as the durable store
- Event Store uses subset of the geteventstore.com functionality
- `https://github.com/rill-event-sourcing/`

Rill Concepts

- Aggregates are reductions of event streams

```
(ns rill.aggregate
  (:require [rill.message :as message]))

(defmulti handle-event
  "Take an event and return the new
state of the aggregate"
  (fn [aggregate event]
    (message/type event)))

(defn update-aggregate
  [aggregate events]
  (reduce handle-event aggregate events))
```

Who & what

Event Sourcing intro

Event Sourcing architecture

How did we get here?

How do we use it

Experiences

Wrapping up

Messages are data

- Commands and events are maps
- Prismatic's Schema for validation & documentation

Example definition

```
(defevent QuestionAnsweredIncorrectly
  :course-id m/Id
  :chapter-id m/Id
  :student-id m/Id
  :question-id m/Id
  :inputs {m/FieldName s/Str}
  chapter-quiz-id)
```

Example event

```
{:rill.message/number 2,
 :rill.message/timestamp
 #inst "2015-02-10T15:19:39.827-00:00",
 :rill.message/id
 #uuid "6f3d10d6-0aac-4ecc-b8c1-a6c297e10c6b",
 :rill.message/type
 :chapter-quiz/QuestionAnsweredIncorrectly,
 :inputs {"_INPUT_1_" "2",
          "_INPUT_2_" "0"},
 :question-id
 #uuid "24f80676-6d1c-4a31-906b-533878270a9b",
 :student-id
 #uuid "2b45e104-821e-4f73-aa15-a5109267214c",
 :chapter-id
 #uuid "1e8d8c4b-0581-4400-bd7d-a66d0500621e"}
```

Handling async/eventual consistency

Show user the effect of *their* actions *now*

- Other user's effect may be propagated slowly
- Command execution returns generated events/number

Standard web pages

- Execute command, redirect to view
- View blocks (refreshes) until event/number is seen

ClojureScript apps

- Execute command, return generated events
- Temporarily refresh global app state

Event Sourcing at Studyflow.nl

Joost Diepenmaat

Who & what

Event Sourcing intro

Event Sourcing architecture

How did we get here?

How do we use it

Experiences

Wrapping up

ClojureScript frontend

- Domain Events are a nice synchronization medium
- SPAs need to deal with eventual consistency
- ClojureScript works nice enough
- Move part of the read-model / querying to the client
- Keep command handling on the server

Who & what

Event Sourcing intro

Event Sourcing architecture

How did we get here?

How do we use it

Experiences

Wrapping up

Experiences

- What works well
- What needs attention

Who & what

Event Sourcing intro

Event Sourcing architecture

How did we get here?

How do we use it

Experiences

Wrapping up

Testing is straightforward

- It's all data!
- Use in-memory store for command and integration tests
- Use plain events for unit testing read-side

Testing commands example

```
(def initial [fixture/course-published-event
              (events/started chapter-id student-id)
              (events/question-assigned
               chapter-id student-id question-id)])
(def expected [(events/question-answered-correctly
                chapter-id student-id question-id
                inputs)
               (events/question-assigned
                chapter-id student-id question-2-id)])
(def command (chapter-quiz/submit-answer!
              course-id chapter-id student-id
              question-id 1 inputs))

(is (command-result= [:ok expected]
                     (execute command initial)))
```

Server performance is great

- Average $<$ 15 ms response time for our busiest application
- Slowest write $<$ 30 ms
- Slowest read $<$ 100 ms (can be improved)

Who & what

Event Sourcing intro

Event Sourcing architecture

How did we get here?

How do we use it

Experiences

Wrapping up

Write logic is constrained

- The system of record and its state is relatively easy to understand
- Things that do go wrong tend to stand out
- Determining aggregate roots is important

Who & what

Event Sourcing intro

Event Sourcing architecture

How did we get here?

How do we use it

Experiences

Wrapping up

You can trust an append-only event stream

- You almost never need to modify written events
- Bugs in the read side of the system have less impact on the write side
- Since you track what's going on you can recover better from bugs
- It's really hard to lose data
- Data-pollution tends to come from runaway processes generating too many events

Multiple views of the same aggregate

- Command view
- Query views
- Front end view for stateful clients

Some of this can possibly be made easier/automated
Some of this is inherent to the CQRS split

- commands need few narrow, shallow aggregates
- queries and views want wide and deep data

Who & what

Event Sourcing intro

Event Sourcing architecture

How did we get here?

How do we use it

Experiences

Wrapping up

Building & adjusting read models

- We use in-memory (non-durable) read models for everything
- Usually straightforward to implement
- Boring & easy to make mistakes (which events apply to this view?)
- Makes deployment slow

Partial performance fixes:

- Sharding & Filtering
- Caching / durable read model

Recommended references

- `http://docs.geteventstore.com/`
- "Event Sourcing Pattern" on MSDN
- "Domain Driven Design" E. Evans
- "Implementing Domain-Driven Design" V. Vernon
- "Patterns of Enterprise Application Architecture" M. Fowler et. al.

Contact

Rill `https://github.com/rill-event-sourcing/`

Studyflow `https://www.studyflow.nl/`

Email joost@studyflow.nl joost@zeekat.nl

Twitter @ZeekatSoftware

We love to talk to you if you're interested in using Rill!

Thanks!

Davide Taviani, Gijs Stuurman, Remco van t Veer, Steven Thonus & Edo van Royen.