An introduction to CQRS and Axon Framework

Finance’s ‘forgotten’ treasure
Allard Buijze – allard.buijze@trifork.nl
Allard Buijze

- Software Architect at Trifork Amsterdam

- ~ 15 years of web development experience

- Strong believer in DDD and CQRS

- Developer and initiator of Axon Framework
  - Java Framework for scalability and performance
  - www.axonframework.org
Layered architecture

- User Interface
- Service Layer
- Data Access Layer

Domain Model
Evolution of a Domain Model

- Customer
- Product
Evolution of a Domain Model

Customer

Order

Product
Evolution of a Domain Model

- ContactDetails
- HelpdeskRequest
- Invoice
- Customer
- Quote
- Order
- Address
- Catalog
- Product
- OrderItem
- Shipment
- Inventory
- PricingStrategy
- DiscountRule
- Location
Evolution of complexity

private static final String PLAYER_COCKPIT_WATERFALL_ITEMS_QUERY =

"select id, " +
    EntityType.NEWS_ITEM.ordinal() + " as entity_type,
    publish_date as sort_date " +
    from news_item
    where active = true and (poster_player_id = :playerId +
    or poster_player_id in (" +
    "select destination_friend_id from friendship where origin_friend_id = :playerId " +
    or project_id in (" +
    "select distinct project_id " +
    from donation " +
    where status = 'OK' +
    or project_id in (" +
    "select distinct project_id from ambassador_project where player_id = :playerId +
    )") +
    or project_id in (" +
    "select distinct project_id from donation where status = 'OK' +
    or project_id in (" +
    "select distinct project_id from ambassador_project where player_id = :playerId +
    )") +
    or raised_via_player_id = :playerId +
    or raised_via_player_id in (" +
    "select destination_friend_id from friendship where origin_friend_id = :playerId " +
    "and " +
    "select destination_friend_id from friendship where origin_friend_id = :playerId " +
    "and destination_friend_id <> :playerId"] +
    )" +
    ) union all (" +
    select id, " +
    EntityType.DONATION.ordinal() + " as entity_type,
    approval_date as sort_date " +
    from donation " +
    where status = 'OK' and (donor_participant_id = :playerId +
    or donor_participant_id in (" +
    "select destination_friend_id from friendship where origin_friend_id = :playerId " +
    or raised_via_player_id = :playerId +
    or raised_via_player_id in (" +
    "select destination_friend_id from friendship where origin_friend_id = :playerId " +
    )") +
    or raised_via_player_id = :playerId +
    or raised_via_player_id in (" +
    "select destination_friend_id from friendship where origin_friend_id = :playerId " +
    "and destination_friend_id <> :playerId"] +
    )" +
    ) union all (" +
    select id, " +
    EntityType.FRIENDSHIP.ordinal() + " as entity_type,
    created as sort_date " +
    from friendship " +
    where origin_friend_id = :playerId or (" +
    "select destination_friend_id from friendship where origin_friend_id = :playerId +
    ) and destination_friend_id <> :playerId"] +
    )" +
    );
Layered architecture

User Interface
- Web Cache
- Session replication

Service Layer
- Method invocation Cache
- Worker pools

Data Access Layer
- Distributed 2nd level cache
- Query Cache

Domain Model
Designed for high performance (?)
Then vs Now

1970’s → 2014

User Interface → Domain Model
Service Layer → Data Access Layer

User Interface → Domain Model
Service Layer → Data Access Layer
Brought to us by the Financial Sector

Sources (ltr ttb): guides.wikinut.com (2x), telegraph.co.uk, commons.wikimedia.org, usatoday.com
Brought to us by the Financial Sector

CQRS

Client

commands

queries

Disruptor (LMAX)

Producer 1

Producer 2

Producer Barrier

Claim Strategy

waitFor(13)

waitFor(12)

waitFor(11)

signal(10)

signal(9)

Consumer Barrier

Wait Strategy

Source: mechanitis.blogspot.com
CQRS Based Architecture

- commands
- queries

Client
CQRS Based Architecture

Command model

commands

Client

queries

Projections
CQRS Based Architecture

Command model

commands

Client

queries

Projections
CQRS Based Architecture

Command model

- T: 1 mln / s
- Resp: < 10 ms

- T: Thr. 20 / s
- Resp: < 100 ms

Projections

- T: 10 mln / s
- Resp. < 100 ms

- T: 1 / s
- Resp. < 10 ms

Client
Synchronizing models

- Command model
- Projections
- Events
- Stored procedures

Client

- Commands
- Queries
CQRS Based Architecture
The power of ubiquitous events

- Reactive
- Cache eviction
- Real-time
- Event Sourcing
- Location transparency
- Systems integration
Event Sourcing
## Event Sourcing

### Orders

<table>
<thead>
<tr>
<th>ID</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Return shipment rcvd</td>
</tr>
</tbody>
</table>

### OrderItems

<table>
<thead>
<tr>
<th>ID</th>
<th>OrderID</th>
<th>Product</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Deluxe Chair</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### VS

<table>
<thead>
<tr>
<th>Seq#</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OrderCreatedEvent</td>
</tr>
<tr>
<td>1</td>
<td>ItemAddedEvent (2x Deluxe Chair - € 399)</td>
</tr>
<tr>
<td>2</td>
<td>ItemRemovedEvent (1x Deluxe Chair - € 399)</td>
</tr>
<tr>
<td>3</td>
<td>OrderConfirmed</td>
</tr>
<tr>
<td>4</td>
<td>OrderCancelledByUserEvent</td>
</tr>
<tr>
<td>5</td>
<td>ReturnShipmentReceived</td>
</tr>
</tbody>
</table>
Event Sourcing

Pros

- Audit trail
- Reconstruct query model(s)
- Management reports since day 1
- Data analysis

Cons

- Maintain history (upcasters)
- Ever-growing
Axon Framework

“CQRS Framework” for Java
- Open source under Apache 2 License

Simplify CQRS based applications
- Provides building blocks for CQRS applications

Current version*: 2.1

More information: www.AxonFramework.org

* On January 9th, 2014
CQRS Based Architecture

UI

Command Handling Component

- command
- persist
- publish
- update
- query

Domain Model

Event Handling Components

Domain Models

Storage

Storage
Axon – Command Bus API

```java
@CommandHandler
public void handle(CreateToDoItemCommand command) {
    // handle command
}

commandBus.dispatch(commandMessage, new VoidCallback() {
    @Override
    public void onSuccess() {
        // query
    }
    @Override
    public void onFailure(Throwable cause) {
        // query
    }
});
```
Axon – Event Bus API

```java
@EventHandler
public void onEvent(ToDoItemCompletedEvent event)
{
    // handle event
}
```

```java
eventBus.publish(asEventMessage(new ToDoItemCompletedEvent("todo1")));
```

![Diagram of Axon Event Bus API](image.png)
CQRS Based Architecture

```
interface AggregateRoot
abstract class AbstractAggregateRoot

interface EventSourcedAggregateRoot
abstract class AbstractAnnotatedAggregateRoot
```
Axon – Event Sourcing

```java
@CommandHandler
public void handle(SeatPlayerCommand command) {
    Participant participant = command.getParticipant();
    if (!getGameState().mayTakeSeat(command.getParticipant())) {
        logInvalidCommand(command);
        return;
    }
    apply(new PlayerSeatedEvent(gameId, getGameState().getDirection(participant)));
    if (getGameState().areAllPlayersSeated()) {
        apply(new RegularGameStartedEvent(gameId, getGameState().getGameDefinition()));
        applyTurnChange();
    }
}

@EventHandler
public void handle(PlayerSeatedEvent event) {
    // update seating state
}

@EventHandler
public void handle(CardPlayedEvent event) {
    // update "cards on table" state
}
```
Event Sourcing - Testing

Given-when-then fixtures
- Given some past events
- When I apply a new Command
- Expect these new Events

```java
fixture.given(new GameStartedEvent(...),
              new CallMadeEvent(...),
              new TurnChangedEvent(...))
    .when(new MakeCallCommand(...))
    .expectEvents(new CallMadeEvent(...),
                  new TurnChangedEvent(...));
```
Separate infrastructure from business logic
Separate infrastructure from business logic
Separate infrastructure from business logic
Spring configuration - Simple

<axon:event-bus id="eventBus"/>

<axon:command-bus id="commandBus"/>
Spring configuration – High performance

```xml
<axon:event-bus id="eventBus"/>

<axon:disruptor-command-bus id="commandBus" event-store="eventStore"
    event-bus="eventBus"
    transaction-manager="transactionManager">
  <axon:repositories>
    <axon:repository id="gameRepository"
        aggregate-type="some.sample.engine.game.RegularGame"/>
  </axon:repositories>
</axon:disruptor-command-bus>
```
Spring configuration – Distributed Events

```xml
<axon:event-bus id="eventBus" terminal="terminal"/>

<axon-amqp:terminal id="terminal" connection-factory="amqpConnection"
                     exchange-name="AxonEventBusExchange">
    <axon-amqp:default-configuration transaction-manager="transactionManager"
                                        transaction-size="25" prefetch="200"
                                        error-handler="loggingErrorHandler"/>
</axon-amqp:terminal>

<axon:cluster id="gameCluster" order="0" default="true">
    <axon:meta-data>
        <entry key="AMQP.Config">
            <bean class="org.axonframework.config.SpringAMQPConsumerConfiguration">
                <property name="queueName" value="GameEngineEvents"/>
            </bean>
        </entry>
    </axon:meta-data>
</axon:cluster>
```
Spring configuration – Distributed Commands

```xml
<bean id="commandBus" class="org.axonframework...DistributedCommandBus">
    <constructor-arg ref="jgroupsConnector"/>
</bean>

<bean id="jgroupsConnector" class="org.axonframework.commandhandling...JGroupsConnectorFactoryBean">
    <property name="serializer" ref="serializer"/>
    <property name="loadFactor" value="${loadFactor:100}"/>
    <property name="localSegment" ref="localCommandBus"/>
    <property name="configuration" value="tcp_gossip.xml"/>
</bean>

<axon:disruptor-command-bus id="localCommandBus" event-store="eventStore"
    event-bus="eventBus"
    transaction-manager="transactionManager">
    <axon:repositories>
        <axon:repository id="gameRepository"
            aggregate-type="some.sample.engine.game.RegularGame"/>
    </axon:repositories>
</axon:disruptor-command-bus>
```
Infrastructure components in Axon

- Single VM
  - SimpleCommandBus
  - SimpleEventBus

- High Performance
  - DisruptorCommandBus
   - ...

- Distributed
  - DistributedCommandBus + JGroupsConnector
  - ClusteringEventBus + AMQP Terminal
   - ...
Axon Roadmap

- More distributed implementations
- Improved OSGi support
- DSL for definition of Command & Events
- IDE Plugins
- High performance Event Store
Axon Framework – Some cases

- **Finance**
  - Process automation in a top 50 bank
  - Trading engine for ETF (index trackers) trading
  - Pension fund calculations at a large bank
  - On-line payment processing

- **Gaming**
  - On-line bridge platform (bridgebig.com)
  - On-line casino (casumo.com)

- **Healthcare**
  - Electronic Medical Record for the Geriatric Healthcare
  - Tracking and Tracing of equipment for dental implants

- **Aviation**
  - Optimizing aircraft movement at a large European airport
The next time…
More information: axonframework.org

Allard Buijze
abu@trifork.com
Please evaluate my talk via the mobile app!