Simon Zambrovski & Jan Galinski - Holisticon AG

Meet AVR0!

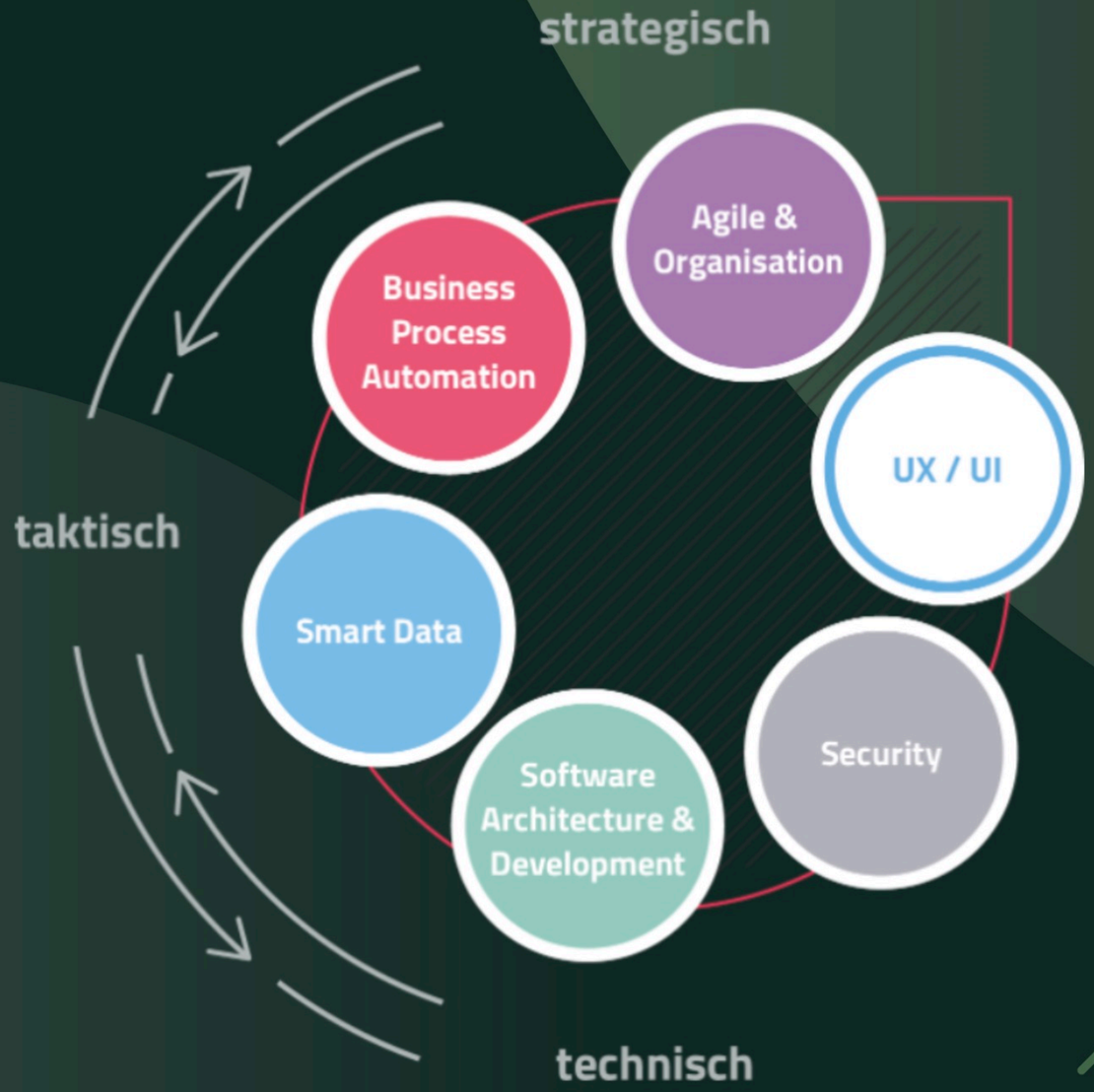
The future of serialization for CQRS/ES systems using Axon Framework.

Who are we?

- Consultants @ holisticon
- Backend developers, software architects
- DDD, CQRS/ES, Axon Framework, Spring, Kotlin, Java

About Holisticon

- IT & Management Consulting
  - Hamburg
  - Hannover
  - Lissabon
- Founded 2007
- **AxonIQ Partner**
- Cool crew (~100 people)
- Kununu Best Employer 2023 in Germany
- And yes ...
  - we are hiring



DDD & CQRS/ES

The Challenges

Business Challenges

Design

- Messages are **first class citizens** *(Event Storming)*
- Your system **behavior** and state is expressed by `commands`, `events` and `queries`
  - `Messages` have **semantics** (a `command` triggers an `event`, a `response` answers a `query`)
- Business and IT need to **communicate** in terms of `messages` and **UBL**
  - Your Business does not work with `String` and `Integer`, it uses `CustomerId` and `Amount`

Evolution

- You will need to **identify** and read `events` years from now
  - `Events` are stored **long term** in an append only store
- If you are forced to change an `event` you need to know if it is still **compatible** to the previous version
- •••
- *You do not want to rely on some developer **hacking** everything correctly in some IDE*

IT Challenges

Development

- Your system will consist of multiple **micro-services** *(polyglot?)*
- You will need to **share** your `commands`, `events` and `queries` between services and teams.
  - You must always use the latest **revisions**
  - You must share **serialization** details as well
- The semantics of your messages and the **axon framework** require you to implement specific handler functions
- You must not make any **mistakes**!

Operation

- No overhead: high **performance** serialization
- Long term storage: You want to have a **compact storage format** that does not waste precious disk space
- If business changes events, you will have to develop smart **upcasting**, keeping compatibility
  - *`Upcasters` need to be **distributed** as well*

QA

- You need to **test** `handlers`, `messages` and `upcasters`

DDD & CQRS/ES

The Solution

What is Apache AVRO?

Self Promotion

Apache Avro™ is the leading serialization format for record data, and first choice for streaming data pipelines.

It offers excellent schema evolution, and has implementations for the JVM (Java, Kotlin, Scala, …), Python, C/C++/C#, PHP, Ruby, Rust, JavaScript, and ***even*** *Perl.*

Fact sheet

- Created in 2011, as part of Hadoop ecosystem
- Platform independent - FE/BE, LLM, Messaging, …
- Schema-First approach for types and protocols
  - Defines a type system *(int, string, long, array, record, …)*
  - Defines serialization encodings *(JSON, Binary, Single Object Encoding)*

What is Apache AVRO?

Example of a simplified bank-account context (json):

```
235
236   "messages": {
237     "createBankAccount": {
238       "doc": "Creates a new bank account.",
239       "meta": {
240         "group": "BankAccountAggregate",
241         "type": "deciderInit"
242       },
243       "request": [
244         {
245           "name": "command",
246           "type": "CreateBankAccountCommand"
247         }
248       ],
249       "response": "BankAccountCreatedEvent",
```

How can Apache AVRO help you?

Design

- **Schema-First** Approach
  - Directly express your `commands`, `events` and `queries` via `schema` declaration
- Describe context behavior as **Protocol**s
  - simple `messages` abstraction for `request`, `response` and `error`
- centralized **Schema-Registry**
- UBL supported by **Logical Types**
  - define custom value types (`CustomerId`, ...)
- Extensible `schema` declaration
  - Express (non)technical properties inside the schema

Evolution

- Schema **compatibility** matrix
  - different levels of forward/backward compatibility
  - separation of `writer` and `reader` **schema**
- Schema **fingerprint** embedded in messages
  - ability to look up the `writer-schema`
- Simplified **upcasting**
  - definition of **default** values for new properties
  - **Conflicts** can be determined on `schema` level
  - The **intermediate representation** *(`GenericRecord`)* understands the underlying `schema` (no mental model mismatch)

How can our AXON–AVRO help?

Development

- Do not share versioned **libraries** between teams, use **schema registry**
- Generate **Core API**
  - (data) classes for `commands`, `events`, `queries`, `responses` and `errors`
- Generate **Command Model**
  - `interfaces` for `CommandHandler` and `EventSourcingHandler` functions
  - custom `CommandGateway` for type safe publishing
- Generate **Query Model**
  - `interfaces` for `QueryHandler` functions
  - `QueryGateway` extensions for type safe and reliant `query/response` clients

Operation

- Pluggable AVRO serializer
  - auto configurable
  - about as fast as `jackson`
- **Compact storage** via `Single Object Encoded` bytes
  - reduce disk usage by min 10% < `json`
- Axon dashboard **plugin** to read events as `json`

QA

- Avoid making any **mistakes** because boilerplate is derived from `schema`!

Demo Time

aka *video time*

TODO

Summary

Summary

What we showed today

- Apache AVRO is a powerful schema first **serialization** platform
- Schema-First is a good idea for DDD/UBL and **Business/IT-Alignment**
- **Code-Generation** improves your life by avoiding boilerplate and bugs
- **Prototype** implementation of `axon-avro` - needs to improve

What we do next

- Implement missing MUST **features**
  - custom logical (value) types
  - registry support
- Explore/implement optional ideas
  - **jMolecules** support - annotated architecture
  - derive **documentation** (context map, plant-uml, markdown, ...)
  - combine with `FModel` - generate delegating aggregate
- Research
  - use it in polyglot environments
  - Data protection and schema?

Thank you!

Links

https://github.com/holixon/axon-avro

https://avro.apache.org/

Questions?

https://holisticon.de/ - https://about.me/zambrovski - https://about.me/jangalinski