Complex event flows in distributed systems

@berndruecker

With thoughts from http://flowing.io
@berndruecker | @martinschimak
3 common hypotheses I check today:

# Events decrease coupling
# Orchestration needs to be avoided
# Workflow engines are painful
Bernd Ruecker
Co-founder and Developer Advocate of Camunda

bernd.ruecker@camunda.com
@berndruecker

Berlin, Germany
ORCHESTRATING A HIGHLY-SCALABLE FULFILLMENT PROCESS

JÖRN HORSTMANN
LUKAS NIEMEIER
2017-03-19
Simplified example: dash button

Photo by 0xF2, available under Creative Commons BY-ND 2.0 license, https://www.flickr.com/photos/0xf2/29873149904/
Three steps...

Pay item → Fetch item → Ship item
Who is involved? Some bounded contexts...
(Micro-)services

- Checkout
- Payment
- Inventory
- Shipment
Autonomous (micro-)services

- Dedicated Application Processes
- Dedicated infrastructure
- Dedicated Development Teams
Events decrease coupling
Example

The button blinks if we can ship within 24 hours
Request/response: temporal coupling

The button blinks if we can ship within 24 hours.
Temporal decoupling with events and read models

*Events are facts about what happened (in the past)
Events can decrease coupling*

*e.g. decentral data-management, read models, extract cross-cutting aspects*
Peer-to-peer event chains

Order placed

Checkout

Payment

Goods fetched

Goods shipped

Inventory

Shipment

Payment received

Checkout

Goods shipped

Payment received

Inventory

Goods fetched

Shipment

Order placed

Checkout

Payment

Goods fetched

Goods shipped

Inventory

Payment received

Inventory

Goods fetched

Goods shipped

Inventory

Payment received

Inventory
Peer-to-peer event chains

Order placed → Checkout → Payment received → Inventory → Goods fetched → Shipment → Goods shipped

checkout
payment
inventory
shipment
order placed
payment received
goods fetched
goods shipped
The danger is that it’s very easy to make nicely decoupled systems with event notification, without realizing that you’re losing sight of that larger-scale flow, and thus set yourself up for trouble in future years.

https://martinfowler.com/articles/201701-event-driven.html
The danger is that it’s very easy to make nicely decoupled systems with event notification, without realizing that you’re losing sight of that larger-scale flow, and thus set yourself up for trouble in future years.

https://martinfowler.com/articles/201701-event-driven.html
The danger is that it’s very easy to make nicely decoupled systems with event notification, without realizing that you’re losing sight of that larger-scale flow, and thus set yourself up for trouble in future years.

https://martinfowler.com/articles/201701-event-driven.html
Peer-to-peer event chains

Order placed

Checkout

Payment

Goods fetched

Goods shipped

Inventory

Shipment

Payment received

Fetch the goods before the payment

Pay item

Fetch item

Ship item
Peer-to-peer event chains

order placed

Checkout

Payment

Inventory

Goods fetched

Goods shipped

Payment received

Fetch the goods **before** the payment

Pay item

Fetch item

Ship item

Goods shipped

Shipment
Peer-to-peer event chains

- Order placed
- Payment received
- Goods fetched
- Goods shipped
- Inventory
- Checkout
- Payment
- Shipment

Fetch the goods **before** the payment

Customers can pay via invoice

...
Extract the end-to-end responsibility

*Commands* have an intent about what needs to happen in the future
Commands help to avoid (complex) peer-to-peer event chains
Orchestration needs to be avoided
Smart ESB-like middleware

- Checkout
- Payment
- Inventory
- Shipment

Order
- Order placed
- Payment received
- Good fetched
- Good shipped
Dumb pipes

Smart endpoints and **dumb pipes**

Martin Fowler
Danger of god services?

A few smart god services tell anemic CRUD services what to do.

Sam Newmann
Danger of god services?

A few smart god services tell anemic CRUD services what to do
A god service is only created by bad API design!
Example

Order → Retrieve Payment → Payment
Example

Order → Retrieve Payment → Payment → Credit Card
Example

Client of **dumb endpoints** easily become a god services.

If the credit card was rejected, the customer can provide new details.
Who is responsible to deal with problems?

If the credit card was rejected, the customer can provide new details.
Who is responsible to deal with problems?

Smart endpoints are potentially long-running

If the credit card was rejected, the customer can provide new details

Clients of **smart endpoints** remains lean.
There was an error while sending your boarding pass
Current situation

Me → Web-UI → Check-in
Current situation

Me → Web-UI → Check-in → Output Mgmt → Barcode Generator
Current situation

Me → Web-UI → Check-in → Output Mgmt → Barcode Generator
Current situation – the bad part
Current situation – the bad part

Me

Stateful Retry

Web-UI

Check-in

Output Mgmt

Barcode Generator
We’re sorry

We are having some technical difficulties at the moment.

Please log on again via www.easyjet.com

If that doesn’t work, please try again in five minutes.

We do actively monitor our site and will be working to resolve the issue, so there’s no need to call.

Go to easyJet.com
We are having some technical difficulties and cannot present you your boarding pass right away.

But we do actively retry ourselves, so lean back, relax and we will send it on time.
Possible situation – much better!
Possible situation – much better!
Possible situation – much better!

The failure never leaves this scope!
Persist thing (Entity, Actor, ...)

State machine or workflow engine

Typical concerns

DIY = effort, accidental complexity

Scheduling, Versioning, operating, visibility, scalability, ...
Workflow engines are painful

complex, proprietary, heavyweight, central, developer adverse, ...
Avoid the wrong tools!

Low-code is great! (You can get rid of your developers!)

Complex, proprietary, heavyweight, central, developer adverse, ...
Workflow engines, state machines

It is relevant in modern architectures

AWS Step Functions
Silicon valley has recognized workflow engines, state machines.
Workflow engines, state machines

There are lightweight open source options

AWS Step Functions

Camunda

Zeebe by Camunda

Conductor

Activiti

JBPM
Workflow engines, state machines

also at scale

Amazon Web Services (AWS) Step Functions

Camunda

Conductor

Netflix OSS

jBPM

Activiti

Uber Cadence

Zeebe by Camunda
Workflow engines, state machines

for today's demo
public static void main(String[] args) {
    ProcessEngine engine = new StandaloneInMemProcessEngineConfiguration().buildProcessEngine();

    engine.getRepositoryService().createDeployment() //
        .addModelInstance("flow.bpmn", Bpmn.createExecutableProcess("flow")) //
        .startEvent() //
        .serviceTask("Step")
        .serviceTask("Step")
        .endEvent() //
        .done() //
    ).deploy();

    engine.getRuntimeService().startProcessInstanceByKey("flow", Variables.putValue("city", "New York"));
}

public class SysoutDelegate implements JavaDelegate {
    public void execute(DelegateExecution execution) throws Exception {
        System.out.println("Hello " + execution.getVariable("city");
    }
}
public static void main(String[] args) {
    ProcessEngine engine = new StandaloneInMemProcessEngineConfiguration()
            .buildProcessEngine();
    engine.getRepositoryService().createDeployment() //
            .addModelInstance("flow.bpmn", Bpmn.createExecutableProcess("flow") //
                .startEvent()
                .serviceTask("Step1").camundaClass(SysoutDelegate.class)
                .serviceTask("Step2").camundaClass(SysoutDelegate.class)
                .endEvent()
            .done()
        ).deploy();

    engine.getRuntimeService().startProcessInstanceByKey(
            "flow", Variables.putValue("city", "New York"));
}

public class SysoutDelegate implements JavaDelegate {
    public void execute(DelegateExecution execution) throws Exception {
        System.out.println("Hello " + execution.getVariable("city"));
    }
}
public static void main(String[] args) {
    ProcessEngine engine = new StandaloneInMemProcessEngineConfiguration()
        .buildProcessEngine();

    engine.getRepositoryService().createDeployment() //
        .addModelInstance("flow.bpmn", Bpmn.createExecutableProcess("flow")
            .startEvent()
            .serviceTask("Step1").camundaClass(SysoutDelegate.class)
            .serviceTask("Step2").camundaClass(SysoutDelegate.class)
            .endEvent()
            .done()
        ).deploy();

    engine.getRuntimeService().startProcessInstanceByKey("flow", Variables.putValue("city", "New York"));
}

public class SysoutDelegate implements JavaDelegate {
    public void execute(DelegateExecution execution) throws Exception {
        System.out.println("Hello " + execution.getVariable("city");
    }
}
```java
public static void main(String[] args) {
    ProcessEngine engine = new StandaloneInMemProcessEngineConfiguration()
            .buildProcessEngine();

    engine.getRepositoryService().createDeployment() //
            .addModelInstance("flow.bpmn", Bpmn.createExecutableProcess("flow")
            .startEvent()
            .serviceTask("Step1").camundaClass(SysoutDelegate.class)
            .serviceTask("Step2").camundaClass(SysoutDelegate.class)
            .endEvent()
            .done())
            .deploy();

    engine.getRuntimeService().startProcessInstanceByKey("flow", Variables.putValue("city", "New York"));
}

public class SysoutDelegate implements JavaDelegate {
    public void execute(DelegateExecution execution) throws Exception {
        System.out.println("Hello " + execution.getVariable("city"));
    }
}
```

**Define flow e.g. in Java DSL**

![Diagram of a BPMN flow with steps labeled Step1 and Step2.]
BPMN
Business Process Model and Notation
ISO Standard
public static void main(String[] args) {
    ProcessEngine engine = new StandaloneInMemProcessEngineConfiguration()
        .buildProcessEngine();

    engine.getRepositoryService().createDeployment() //
        .addModelInstance("flow.bpmn", Bpmn.createExecutableProcess("flow")
            .startEvent()
            .serviceTask("Step1").camundaClass(SysoutDelegate.class)
            .serviceTask("Step2").camundaClass(SysoutDelegate.class)
            .endEvent()
            .done()
        ).deploy();

    engine.getRuntimeService().startProcessInstanceByKey(
        "flow", Variables.putValue("city", "New York"));
}

public class SysoutDelegate implements JavaDelegate {
    public void execute(DelegateExecution execution) throws Exception {
        System.out.println("Hello " + execution.getVariable("city");
    }
}
public static void main(String[] args) {
    ProcessEngine engine = new StandaloneInMemProcessEngineConfiguration()
        .buildProcessEngine();

    engine.getRepositoryService().createDeployment() //
        .addModelInstance("flow.bpmn", Bpmn.createExecutableProcess("flow")
            .startEvent()
            .serviceTask("Step1").camundaClass(SysoutDelegate.class)
            .serviceTask("Step2").camundaClass(SysoutDelegate.class)
            .endEvent()
        .done()
    ).deploy();

    engine.getRuntimeService().startProcessInstanceByKey(
        "flow", Variables.putValue("city", "New York"));
}

public class SysoutDelegate implements JavaDelegate {
    public void execute(DelegateExecution execution) throws Exception {
        System.out.println("Hello "+ execution.getVariable("city"));
    }
}
public static void main(String[] args) {
    ProcessEngine engine = new StandaloneInMemProcessEngineConfiguration()
        .buildProcessEngine();

    engine.getRepositoryService().createDeployment() //
        .addModelInstance("flow.bpmn", Bpmn.createExecutableProcess("flow")
        .startEvent()
        .serviceTask("Step1").camundaClass(SysoutDelegate.class)
        .serviceTask("Step2").camundaClass(SysoutDelegate.class)
        .endEvent()
    .done()
).deploy();

    engine.getRuntimeService().startProcessInstanceByKey("flow", Variables.putValue("city", "New York"));
}

public class SysoutDelegate implements JavaDelegate {
    public void execute(DelegateExecution execution) throws Exception {
        System.out.println("Hello " + execution.getVariable("city"));
    }
}
Now you have a state machine!

Charge credit card

Rejected

Inform customer of rejection

Wait for customer to update card

Payment received

7 days

Payment failed

Payment
Easy to handle time
Stateful retries

- Payment
  - REST
  - Credit Card

Charge credit card

Stateful retry: e.g. 10 times, delay 15 min.
Fallbacks increase resilience

Charge credit card (Provider A) → Payment received → Charge credit card (Provider B) → Payment received

Payment → REST → Credit Card
Distributed systems
It is impossible to differentiate certain failure scenarios. Independent of communication style!
Distributed systems introduce complexity you have to tackle!
Distributed systems introduce complexity you have to tackle!
Life beyond Distributed Transactions: 
an Apostate’s Opinion
Position Paper

Pat Helland
Amazon.Com
705 Fifth Ave South
Seattle, WA 98104
USA

Email: PHelland@Amazon.com

ABSTRACT

Many decades of work have been invested in the design and refinement of protocols such as 2PC.  Poorly designed transactions often make business processes more complex, rather than simpler.  Instead, applications are built using different techniques which do not provide the same transactional guarantees but still meet the needs of their businesses.

This paper explores

...
Distributed transaction with compensations* 

*aka Saga-Pattern
Living documentation for long-running behaviour
Visual HTML reports for test cases
Proper Operations

Visibility + Context
Before mapping processes explicitly with BPMN, the truth was buried in the code and nobody knew what was going on.

Jimmy Floyd, 24 Hour Fitnesse
Workflows live inside service boundaries
Explicit flows help separate domain and infrastructure

Aggregates, Domain Events, Domain Services, etc ...

+ the flow
Manigfold architecture options

https://blog.bernd-ruecker.com/architecture-options-to-run-a-workflow-engine-6c2419902d91
Manigfold architecture options

https://blog.bernd-ruecker.com/architecture-options-to-run-a-workflow-engine-6c2419902d91
Manigfold architecture options

https://blog.bernd-ruecker.com/architecture-options-to-run-a-workflow-engine-6c2419902d91
Manigfold architecture options

Architecture options to run a workflow engine

This week a customer called and asked (translated into my own words and shortened):

“We do composite services, orchestrating two or three CRUD services to do something more useful. Our architects want to use your workflow engine for this because the orchestration flow might be long running. Is this a valid scenario for the workflow engine? Currently we run one big central cluster for the workflow engine—won’t that get a rest?”

These are valid questions which recently we get asked a lot. Sometimes in the context of microservices modern apps require an architecture that is not always aligned.
Manigfold architecture options

https://blog.bernd-ruecker.com/architecture-options-to-run-a-workflow-engine-6c2419902d91
Lightweight workflow engines are great — don’t DIY*

*e.g. enabling potentially long-running services, solving hard developer problems, can run decentralized
Zalando

Sales-Order and Order-Fulfillment via Camunda for every order worldwide


Sales 2016: 3,6 Mrd. EUR
Growth 2016: 23%
Code example & live demo

Checkout
Order
Payment
Inventory
Shipping
Monitor

Human Tasks

kafka

https://github.com/flowing/flowing-retail/
# Events decrease coupling: sometimes
read-models, but no complex peer-to-peer event chains!

# Orchestration needs to be avoided: sometimes
no ESB, smart endpoints/dumb pipes, important capabilities need a home

# Workflow engines are painful: some of them
lightweight engines are easy to use and can run decentralized,
they solve hard developer problems, don’t DIY
Contact: bernd.ruecker@camunda.com
@berndruecker

Slides: https://bernd-ruecker.com

Blog: https://blog.bernd-ruecker.com

Code: https://github.com/flowing


https://www.infoq.com/articles/events-workflow-automation

With thoughts from http://flowing.io
@berndruecker | @martinschimak