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

JÖRHN HÖRSTMANN
LUKAS NIEMEIER

2017-05-19
Simplified example: dash button
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
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)

The button blinks if we can ship within 24 hours
Events can decrease coupling*

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

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

Actions:
- Pay item
- Fetch item
- Ship item
Peer-to-peer event chains

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

Checkout

Inventory

Payment

Shipment

Goods shipped

Payment received

Goods fetched
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

Goods fetched

Payment

Goods shipped

Inventory

Goods fetched

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
- Shipment
- Goods fetched
- Goods shipped
- Payment received

Fetch the goods before the payment

Customers can pay via invoice

...
Peer-to-peer event chains

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

Fetch the goods before the payment
Peer-to-peer event chains

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

Fetch the goods **before** the payment.
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

Order
- order placed
- payment received
- good fetched
- good shipped
Dumb pipes

Checkout

Order

Payment

Shipment

Inventory

Smart endpoints and **dumb pipes**

Martin Fowler
Danger of god services?

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

- Payment
- Inventory
- Order
- Shipment

Sam Newmann
Danger of god services?

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

Sam Newmann

Checkout
Order
Payment
Shipment
Inventory
A god service is only created by bad API design!
Example

Order → Retrieve Payment → Payment
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.

Payment received, but failed.
Clients of **smart endpoints** remains lean.

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

Persist thing
(Entity, Document, 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!)

Death by properties panel

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

It is relevant in modern architectures
Silicon valley has recognized workflow engines, state machines.
Traditionally, some of these processes had been orchestrated in an ad-hoc manner using a combination of pub/sub, making direct REST calls, and using a database to manage the state. However, as the number of microservices grow and the complexity of the processes increases, getting visibility into these distributed workflows becomes difficult without a central orchestrator.
Workflow engines, state machines

There are lightweight open source options

AWS Step Functions

Camunda

Activiti

jBPM
Workflow engines, state machines

also at scale

AWS Step Functions

Camunda

Conductor

Activiti

Netflix OSS

jBPM
How we built a highly scalable distributed state machine

Say hello to “big workflow”—part 2

In Zeebe.io—a highly scalable distributed workflow engine—I described that Zebee is a super performant, highly scalable and resilient workflow engine. I described that this allows to leverage workflow automation in a lot more use cases, allowing scenarios. I revealed that Zebee can be used with Kafka. I revealed that...
public static void main(String[] args) {
    ProcessEngine engine = new StandaloneInMemProcessEngineConfiguration()
        .buildProcessEngine();

    engine.getRepositoryService().createDeployment() //
        .addModelInstance("flow.bpmn", Bpmn.createExecutableProcess("flow") //
            .startEvent() //
            .serviceTask("Step1") //
            .serviceTask("Step2") //
            .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"));
}

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"));
    }
}
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()
        ).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!
Easy to handle time
Atomic vs. composite command execution

Typically we see a "command" as the intent to change a write model...

... but the customer's or service clients intent is often targeted at a more valuable business result, which needs many steps to be achieved.

Place order  Order placed  Atomic, trans-actional execution

Place order  Order fulfilled  Composite, long-running execution
Synchronous communication
Synchronous communication
Synchronous communication
Synchronous communication

Order → Payment → Credit Card

Payment → REST

Stateful Retry

Insertion: Stateful retry: e.g. 10 times, delay 15 min.

Charge 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!
2007

Life beyond Distributed Transactions: an Apostate’s Opinion

Position Paper

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

The positions expressed in this paper are personal opinions and do not in any way reflect the positions of my employer, Amazon.com.

ABSTRACT

Many decades of work have been invested in the area of distributed transactions, mainly protocols such as 2PC. Current approaches to ensuring the correct execution of transactions in the face of failures require that the participating processes maintain global states and communicate frequently with each other. These approaches are fine for ensuring that the correct outcomes of transactions are available to the users of applications, but they are expensive in terms of the amount of communication and storage required. Many applications, especially in the enterprise arena, do not expect these guarantees and are built using different techniques which do not provide the same transactional guarantees but still meet the needs of their businesses.

This paper explores these approaches and the problems that have been encountered using them.
Distributed transactions using compensation

Payment retrieval requested → Deduct existing customer credit

Security check → Charge credit card

Payment complete? → Restore customer credit

Payment received → Compensation

Credit card failed → Payment failed
Relaxed consistency

- Temporarily inconsistent state
- But eventually consistent
- No Isolation (as in ACID)
5 Workflow Automation Use Cases You Might Not Have Considered

https://thenewstack.io/5-workflow-automation-use-cases-you-might-not-have-considered/
Use cases for workflow automation
Biz Dev Ops

- improve communication
- improve communication
Biz Dev Ops

Leverage state machine & workflow engine

Living documentation

Visibility in testing
Visual HTML reports for test cases
Biz Dev Ops

Understand and discuss business processes

Evaluate optimizations in-sync with implementation

Leverage state machine & workflow engine

Living documentation

Visibility in testing

improve communication

improve communication
Living documentation for long-running behaviour

1. **Payment required**
   - Use existing customer credit
     - Payment complete?
       - Yes: Charge credit card
         - Payment received
       - No: Credit card expired
         - Wait for customer to update credit card
           - Two weeks
             - Payment failed
   - Restore former customer credit

2. **Payment failed**
   - Charge credit card
   - Credit card expired
     - Wait for customer to update credit card
       - Two weeks
         - Payment failed
Biz Dev Ops

- Understand and discuss business processes
- Evaluate optimizations in-sync with implementation
- Leverage state machine & workflow engine
- Improve communication
- Living documentation
- Visibility in testing
Biz Dev Ops

Understand and discuss business processes
Evaluate optimizations in-sync with implementation

Leverage state machine & workflow engine
Living documentation

operate with visibility and context

Visibility in testing

improve communication
improve communication

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
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
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 our workflow engine for this because the orchestration flow might be long running. Is this a valid scenario for the workflow? Currently we run one big central cluster for the workflow engine—won’t that get a nose?”

These are valid questions which recently we get asked a lot. Consider the context of microservices, modern data ideas.
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 workflow? Currently we run one big central cluster for the workflow engine—won’t this get a nosq?”

These are valid questions which recently we get asked a lot. Customers in the context of microservices modern DevOps model.
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 workflow? Currently we run one big central cluster for the workflow engine — won’t that get a nope?”

These are valid questions which recently we got asked a lot, especially in the context of microservices, modern API management, and

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
Sales-order & order-Fulfillment
via Camunda
for every order worldwide
(Q2 2017: 22,2 Mio)
Code example & live demo

- Checkout
- Order
- Payment
- Inventory
- Shipping
- Monitor
- Human Tasks

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
Thank you!
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

https://thenewstack.io/5-workflow-automation-use-cases-you-might-not-have-considered/

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