COPPER

- Next Generation SPOCK -

13.10.2011

Michael Austermann
SCOOP Software GmbH

Agenda

- Motivation
  - Yet Another Workflow Engine?
- Anforderungen an eine Workflow Engine
- COPPER
- Aussicht
- Fazit

Grundproblem

- Anforderungen
  - Langlaufende Geschäftsprozesse
    - Stunden, Tage, Wochen, Monate
  - Crash-Safety
  - Hoher Durchsatz
- Gegebenheiten
  - Gängige Programmiersprachen bzw. deren Laufzeit-Umgebungen sind i.d.R. transient
    - Java, C++, C#, perl
  - Kontrollfluß/Stack ↔ Thread

Sprachen & Notationen

- Spezielle Sprachen und Notationen
  - Standards
    - BPEL - Business Process Execution Language
      - XML/WS basiert
    - XPDL – XML Process Definition Language
    - BPMN - Business Process Modelling Notation
  - Proprietär
    - SPOCK
      - Inifile-Syntax
      - XML-Syntax
    - SAG webMethods
      - Trennung Process-Designer & -Developer

SPOCK - inifile

```
[GRAPH-fup-de-resetbwm-1]
  description = Workflow for the <b>FUP-DE</b> programme for bandwidth modification
overriding.

  RequestFactory = de.scoopgmbh.rmc.mediator.riskproc.request.BatchIERORequestFactory

  command-10 = NOP
  command-10.start = true
  command-10.ec.all = BATCHED_CHECKPOINT
  command-10.0.goto-20 = default

  command-20 = REPLY
  command-20.0.goto-300 = default

  command-300 =
de.scoopgmbh.rmc.mediator.riskproc.command.batch.GetAndCheckCustomerCmd$Factory
  command-300.checkEligibility = false
  command-300.result = java.lang.String
  command-300.0.goto-5000 = not (result = "OK")
  command-300.1.goto-400 = default

  command-400 =
de.scoopgmbh.rmc.mediator.riskproc.command.common.GetAndCheckC2PCmd$Factory
  command-400.result = java.lang.String
  command-400.0.goto-3000 = $${request.reject}
  command-400.1.goto-5000 = not (result = "OK")
  command-400.2.goto-500 = $${request.c2pRegData.firstUse == null}
  command-400.3.goto-550 = default
```

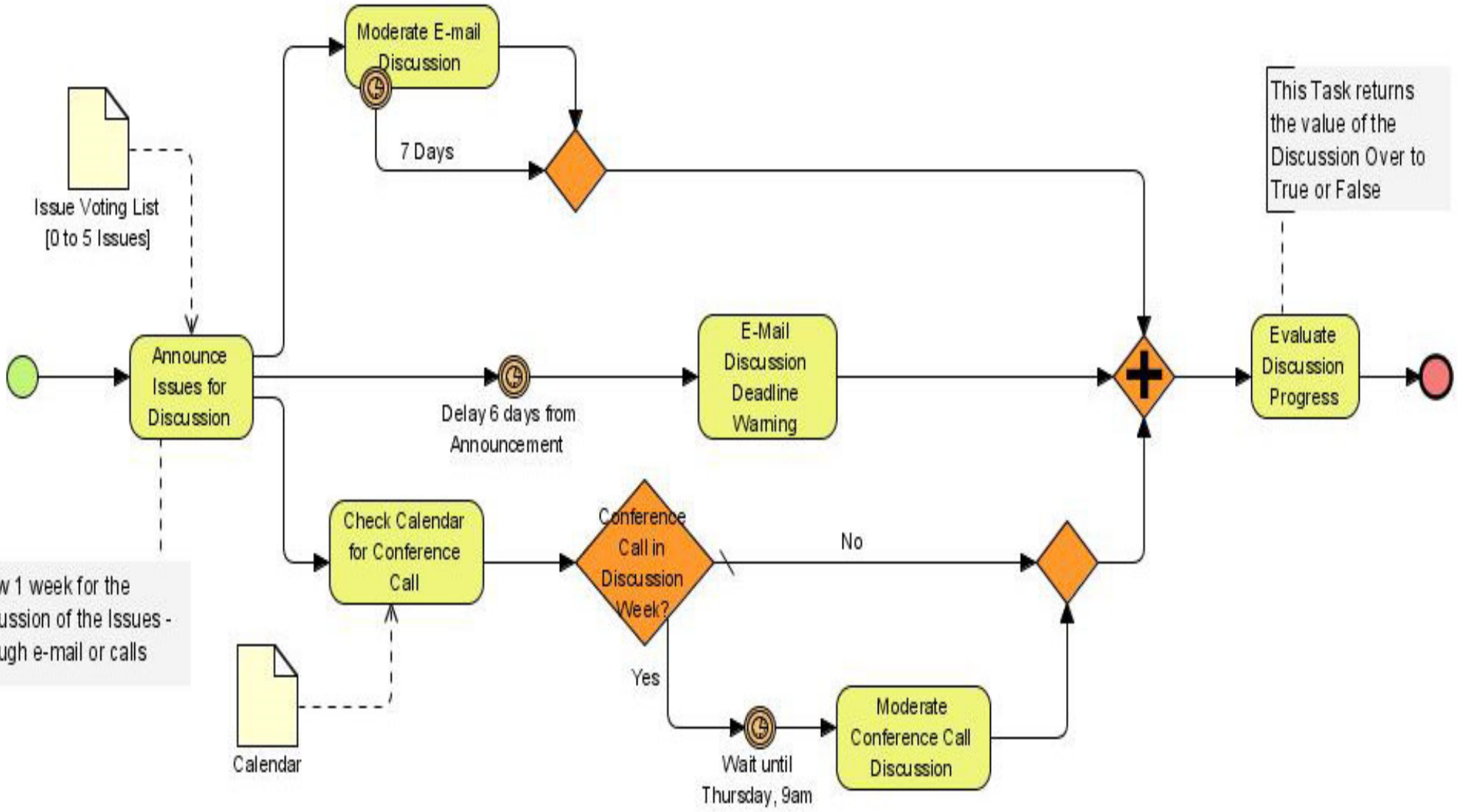
SPOCK - XML

```
<?xml version="1.0" encoding="UTF-8"?>
<ns:graph id="regression-1.0" deleteOnCompletion="false" xmlns:ns="
http://scoopgmbh.de/spock/graph/xml">

  <ns:property name="NUMBER_OF_CP_TO_EXEC">19</ns:property>

  <ns:command label="10" factory="de.scoopgmbh.spock.commands.special.NOPCommand$Factory"
    start="true">
    <ns:ec>
      <ns:all>BATCHED_CHECKPOINT</ns:all>
    </ns:ec>
    <ns:successor target="11">true</ns:successor>
  </ns:command>

  <ns:command label="11" factory="jsr">
    <ns:successor target="1000">target</ns:successor>
    <ns:successor target="12">true</ns:successor>
  </ns:command>

  <ns:command label="12"
        factory="de.scoopgmbh.spock.commands.special.AcquirePersistentLockCommand$Factory"
    pool="HIGHPRIO">
    <ns:ec>
      <ns:all>CHECKPOINT</ns:all>
    </ns:ec>
    <ns:property name="MUTEX">$${request.mutex}</ns:property>
    <ns:property name="INTERVAL">30000</ns:property>
    <ns:successor target="15">true</ns:successor>
  </ns:command>
```

BPEL


```
<process name="BusinessTravelProcess">
  <partnerLinks>
    <partnerLink name="client"
                 partnerLinkType="trv:travelLT"
                 myRole="travelService"
                 partnerRole="travelServiceCustomer"/>
  </partnerLinks>
  <variables>
    <variable name="TravelResponse" messageType="aln:TravelResponseMessage"/>
  </variables>
  <sequence>
    <receive partnerLink="client"
             portType="trv:TravelApprovalPT"
             operation="TravelApproval"
             variable="TravelRequest"
             createInstance="yes" />
    <assign>
      <copy>
        <from variable="TravelRequest" part="employee"/>
        <to variable="EmployeeTravelStatusRequest" part="employee"/>
      </copy>
    </assign>
    <invoke partnerLink="employeeTravelStatus"
            portType="emp:EmployeeTravelStatusPT"
            operation="EmployeeTravelStatus"
            inputVariable="EmployeeTravelStatusRequest"
            outputVariable="EmployeeTravelStatusResponse" />
...
```

BPMN


Moderate E-mail Discussion
7 Days
Issue Voting List
[0 to 5 Issues]
This Task returns the value of the Discussion Over to True or False
Announce Issues for Discussion
Delay 6 days from Announcement
E-Mail Discussion Deadline Warning
Evaluate Discussion Progress
Check Calendar for Conference Call
Conference Call in Discussion Week?
No
Yes
Allow 1 week for the discussion of the Issues - through e-mail or calls
Calendar
Wait until Thursday, 9am
Moderate Conference Call Discussion

Nachteile spezieller Sprachen

- Einarbeitungsaufwand (Sprache, Notation, Tools, Laufzeit-Umgebung, Monitoring)
- Skill-Verfügbarkeit
- Lizenz-Kosten
- Für „Anwender“ zu unverständlich, für „Programmierer“ zu umständlich
- „Einfache Dinge einfacher, komplizierte Dinge komplizierter“

Wunsch

- Einfache Workflow-Engine mit
  - Einfacher oder allgemein bekannter Sprache/Notation,
  - hoher Performance,
  - einfachem Ausführungsmodell bzw. einfacher Laufzeitumgebung.

...

- JAVA als Sprache für Workflows / Geschäfts-prozesse
- … mit SPOCK-ähnlicher Laufzeitumgebung
  - Prozessor-Pools
  - DB Batching
  - Asynchronous Continuations

Problem

- Enge Kopplung Stack <==> Thread
- Kontrollfluss nicht an beliebiger Stelle unterbrechbar UND wiederausführbar

Mögliche Lösungswege

- Open Source Projekte
  - Continuations
    - Jetty: Gekoppelt an HTTP container
    - Apache Javaflow
    - Andere: Alpha/Beta
- SCOOP
  - JavaJavaVM - Bytecode Interpreter für Java
  - Bytecode Instrumentierung
    - BCEL
    - ASM

COPPER

- Bytecode Instrumentierung (mittels ASM Library)
- Wait/Interrupt und Resume an beliebiger Stelle
  - Call Stack wird gesichert und wiederhergestellt

Beispiel Workflow

```
public class VerySimpleWorkflow extends Workflow<String> {

    private MockAdapter mockAdapter;

    @AutoWire
    public void setMockAdapter(MockAdapter mockAdapter) {
        this.mockAdapter = mockAdapter;
    }

    @Override
    public void main() throws InterruptException {
        System.out.println("started");
        for (int i=0; i<3; i++) {
            final String cid = mockAdapter.foo("foo");
            System.out.println("about to wait...");
            waitForAll(cid);                    // COPPER - WAIT & RESUME
            System.out.println("resumed");
            Response r = getAndRemoveResponse(cid);
        }
        System.out.println("finished");
    }
}
```

Beispiel MOCK Adapter

```
public class MockAdapter {

  private ProcessingEngine engine;

  // ...

  public String foo(String data) {
    // Apache CXF asynchronous webservice invokation

    final String correlationId = engine.createUUID(); // COPPER
    fooService.fooAsync(data, new AsyncHandler() {
        public void handleResponse(Response r) {
          engine.notify(r, correlationId); // COPPER
        }
    });
    return correlationId;
  }
}
```

Java Code Beispiel


```
public class InterruptableWorkflow extends ...{

    @Override
    public void main() throws InterruptException {
        System.out.println("started");
        //...
        System.out.println("interrupting now");
        super.interrupt();
        System.out.println("resumed");
        // ...
        System.out.println("finished");
    }
}
public class Main {
    public static void main(String[] args) {
        InterruptableWorkflow w = new InterruptableWorkflow();
        for (;;) {
            try {
                w.main();
                // not interrupted
                break;
            }
            catch(InterrupException e) {
                // interrupted
            }
        }
    }
}
```

Einschränkungen

- Aktuell noch Einschränkungen
  - wait/interrupt nur innerhalb der Klassen-Hierarchie mit Workflow als Basis-Klasse
- Wird in späteren Versionen jedoch möglich sein

COPPER Instrumentierung

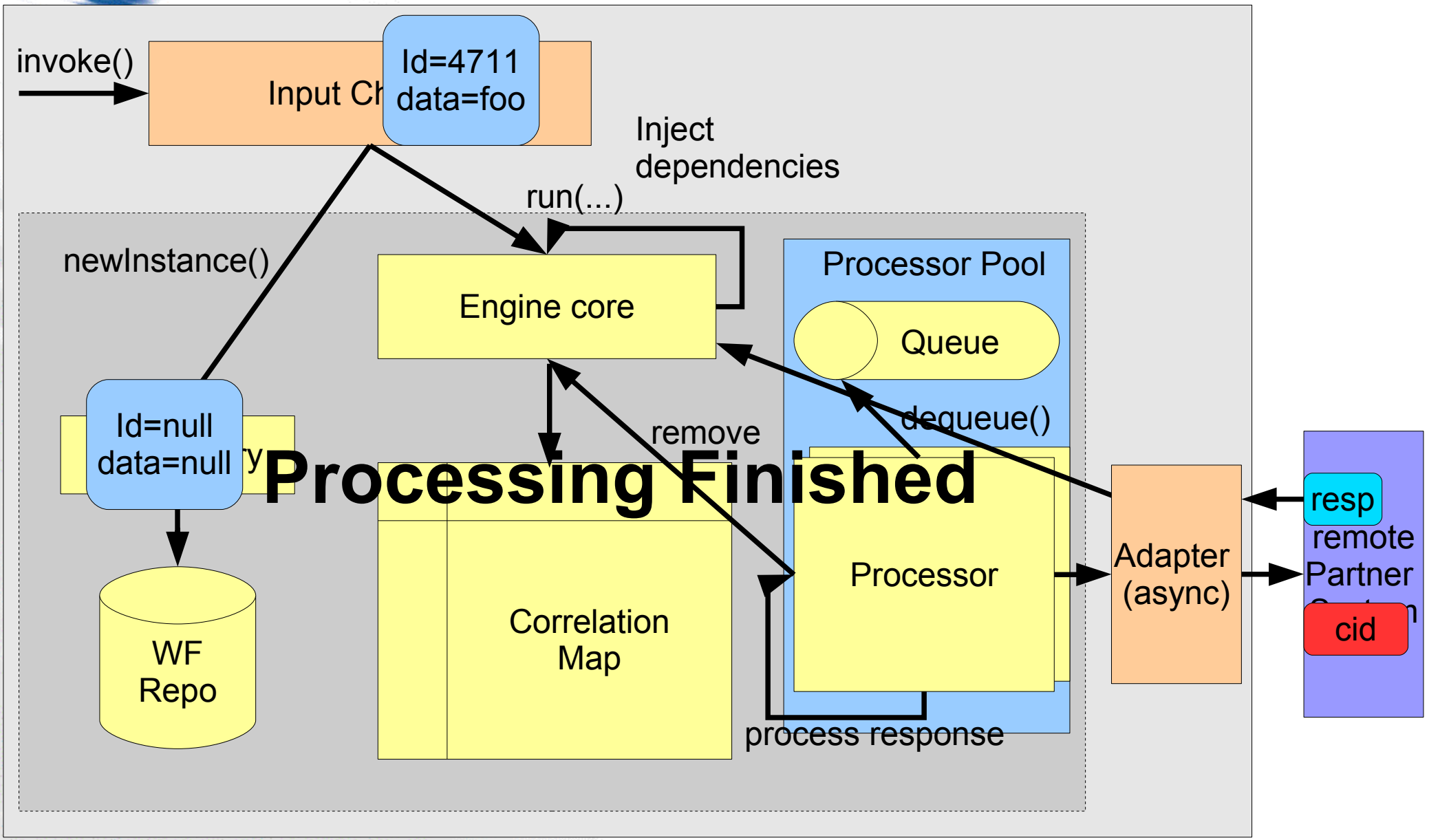
- Bytecode Instrumentierung (ASM)
- Interrupt und Resume an beliebiger Stelle

- Entkoppelt Call-Stack und Thread
- Ermöglicht Persistenz von Call-Stacks
- Basis-Technologie in COPPER

COPPER

- Transiente und persistente Workflows
- SPOCK ähnlicher Architektur
  - Processor Pools
  - Queues
  - …
- JAVA als Kern-Beschreibungssprache
- Weitere Sprachen „on top“ möglich
  - BPEL, BPMN, ...

Ausführung


invoke()
Input Ch
Id=4711
data=foo
Inject
dependencies
run(...)
newInstance()
Engine core
Processor Pool
Queue
dequeue()
Id=null
data=null
remove
Processing Finished
WF
Repo
Correlation
Map
Processor
Adapter
(async)
resp
remote
Partner
cid
process response

Weitere COPPER Features

- Umfassendes Response-Handling
  - Early Responses möglich
  - Multiple Responses möglich (*first* oder *all*)
  - Beliebige CorreleationId
- Parallelität durch gleichzeitige Partner System calls
- Beliebiger Container, z.B. Spring, Inifile
- Dependency Injection für Workflow-Instanzen
- Mehrere Engine Instanzen in einer JVM
- Workflow dynamisch zur Laufzeit änderbar

Weitere COPPER Features der Persistent Engine

- Keine Queue-Overflows
  - solange DB-Storage ausreicht
- Verteilte Ausführung auf n Engines (Milest. 1.2)
  - Load Balancing
  - Redundanz
  - Ausfallsicherheit
    - Benötigt HA DBMS, z.B. Oracle RAC
- Audit Trail

Transient Engine

- Performance
  - Dell Studio XPS 1640
    - Intel Core2 Duo CPU P8600 2,40 GHz
    - 8 GB RAM
    - 5,3 Windows-Leistungsindex
  - Windows Vista 64bit
- Workflow mit 10 asynchronen Invokes
- ~ **240.000** wait/notify pro Sekunde
  - Entspricht 240.000 Async SPOCK Commands
  - ~ 1,5 x schneller als SPOCK

Persistent Engine

- Performance
  - COPPER Engine auf Studio XPS
  - DB auf proliant (Oracle 11g2)
- Workflow mit 10 asynchronen Invokes
- ~ 6.000 wait/notify pro Sekunde
  - 600 BPs/Sekunde
- ~ ungefähr so schnell wie SPOCK

Pitfalls

- „data“ member und alle lokalen Variablen in persistenten Workflows müssen Serializable sein.
- Kein COPPER „wait“ in synchronized Blöcken
- Keine Delegation zwischen Workflow Instanzen

Aussicht

- Monitoring (technisch und BAM)
  - JMX
- CAMEL Integration
- Schnelles Housekeeping (in Kürze)
- Open Source Veröffentlichung
  - Umstellung auf ant/ivy
  - Tabellen-Umbenennungen
  - Oracle-JDBC Abhängigkeit
  - License Prefix (Apache 2.0)

Fazit

- Leichtgewichtige und performante Engine
- Java als Beschreibungs-Sprache