Web Services, Orchestration and Apache Ode

Alex Boisvert, Intalio Inc.

ApacheCon EU 2008

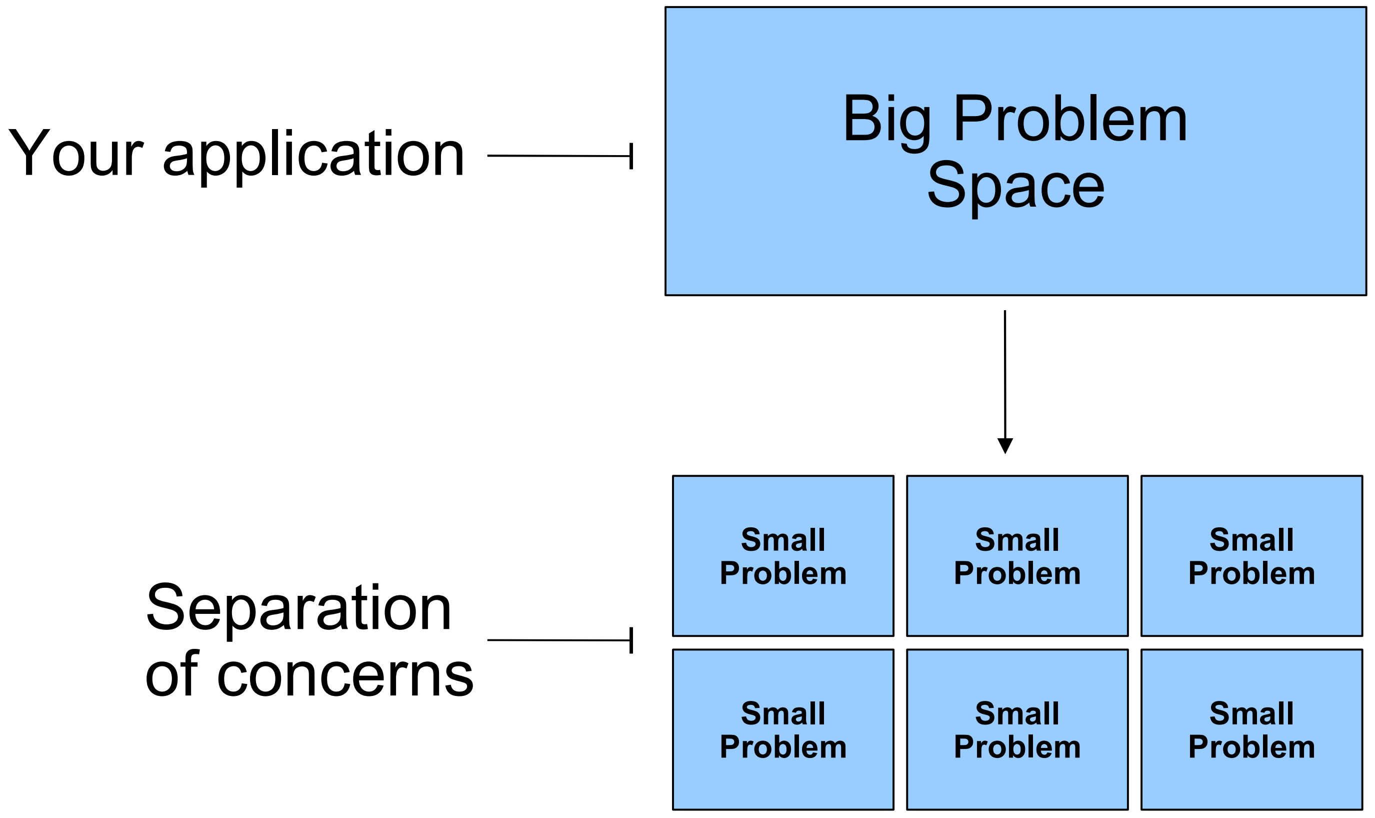
Outline

- Overview of BPEL
- Apache Ode
- Best Practices
- What's Coming

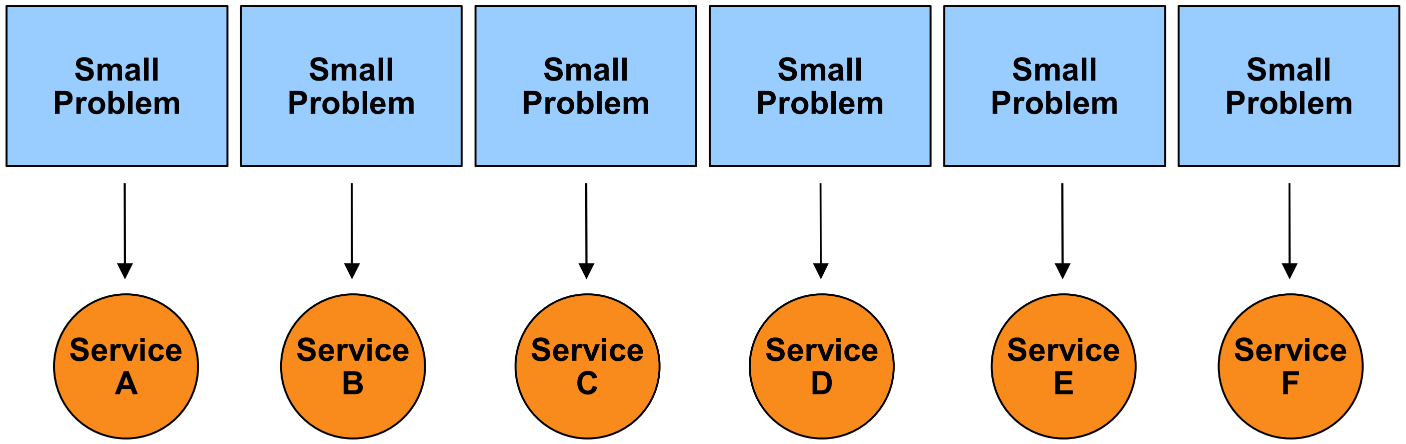
SOA in 3 Minutes

- Key design principles
  - Standardized service contract
  - Service abstraction
  - Loose-coupling
  - Reusability
  - Autonomy
  - Statelessness
  - Composability

Step 1: Divide and Conquer



Step 2: Create Services

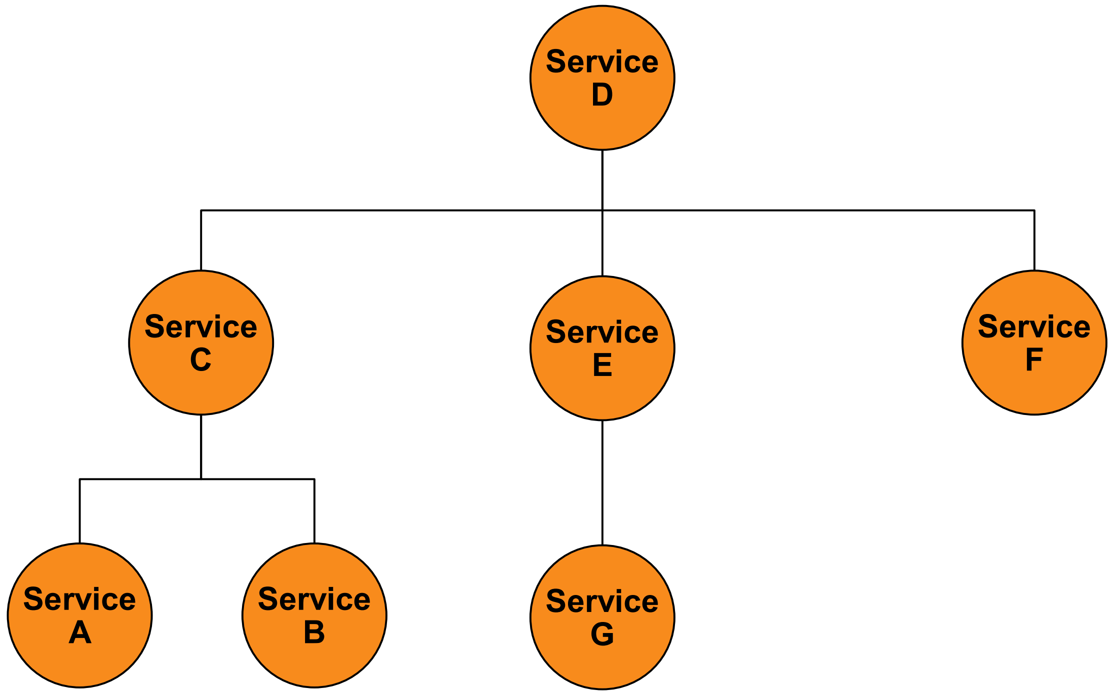


Composable services

(Loosely coupled? Reusable? Autonomous? Stateless?)

Step 3: Composition

Composite Application



Solves the Big Problem

So far so good,
but where are the
business processes?

How do you separate
process concerns
from the rest?

Enter BPEL

- What is BPEL?
  - A programming language for specifying business process within a service-oriented architecture... **as a separate concern**
- Benefits
  - Quickly compose services into new applications
  - Easily extend applications from the outside
  - Agile adaptation to your changing business
  - Clearer view of processes helps manage processes at the business level

BPEL Goals

- Specifying business process behavior
  - Executable Processes
  - Abstract Processes
- Programming in the large
  - Long-running processes
  - Transactions and compensation
  - Message and instance correlation
  - Leverage web services

What is it based on?

- Web service description
  - WSDL 1.1
- Data Model
  - XML Schema 1.0, Infoset
  - XPath 1.0, XSLT 1.0
- Flow Control
  - Hierarchical structure
  - Graph-based flows

Overview of BPEL Activities

- Message exchange
  - <invoke>
  - <receive> ... <reply>
- Data manipulation
  - <assign>
- Control flow
  - <if> ... <else>
  - <while>
  - <repeatUntil>

Overview BPEL Activities

- Parallel and graph-based processing
  <forEach>, <flow> with <link>'s
- Event processing and timers
  <pick>, <onEvent>, <onMessage>, <onAlarm>
- Exception and error handling
  <throw>, <catch>, <rethrow>, <compensate>
- Miscelaneous
  <wait>, <empty>, <validate>, ...

BPEL is **not**

- Not a general-purpose programming language
  - But a great complement to your existing general-purpose programming language(s)
- Not a human workflow language
  - But it ***can*** support human workflow via extensions (BPEL4People) or integration with workflow web services
- Not a graphical notation
  - But there's a great standard notation (BPMN) and many great tools for graphical modeling

BPEL is **not**

- Not limited to business processes
  - Used in grid computing, SOA testing, automation, etc.
- Not limited to WSDL or XML
  - Extensible type and expression system
  - Connect to legacy systems, REST services, invoke Java objects, send files via FTP, JMS publish/subscribe, ...
  - Whatever your ESB can do

BPEL is **not**

- Not meant to be hand-written
  - XML syntax chosen for interoperability* between tools (import/export)
  - You should use tools
  - Seriously.

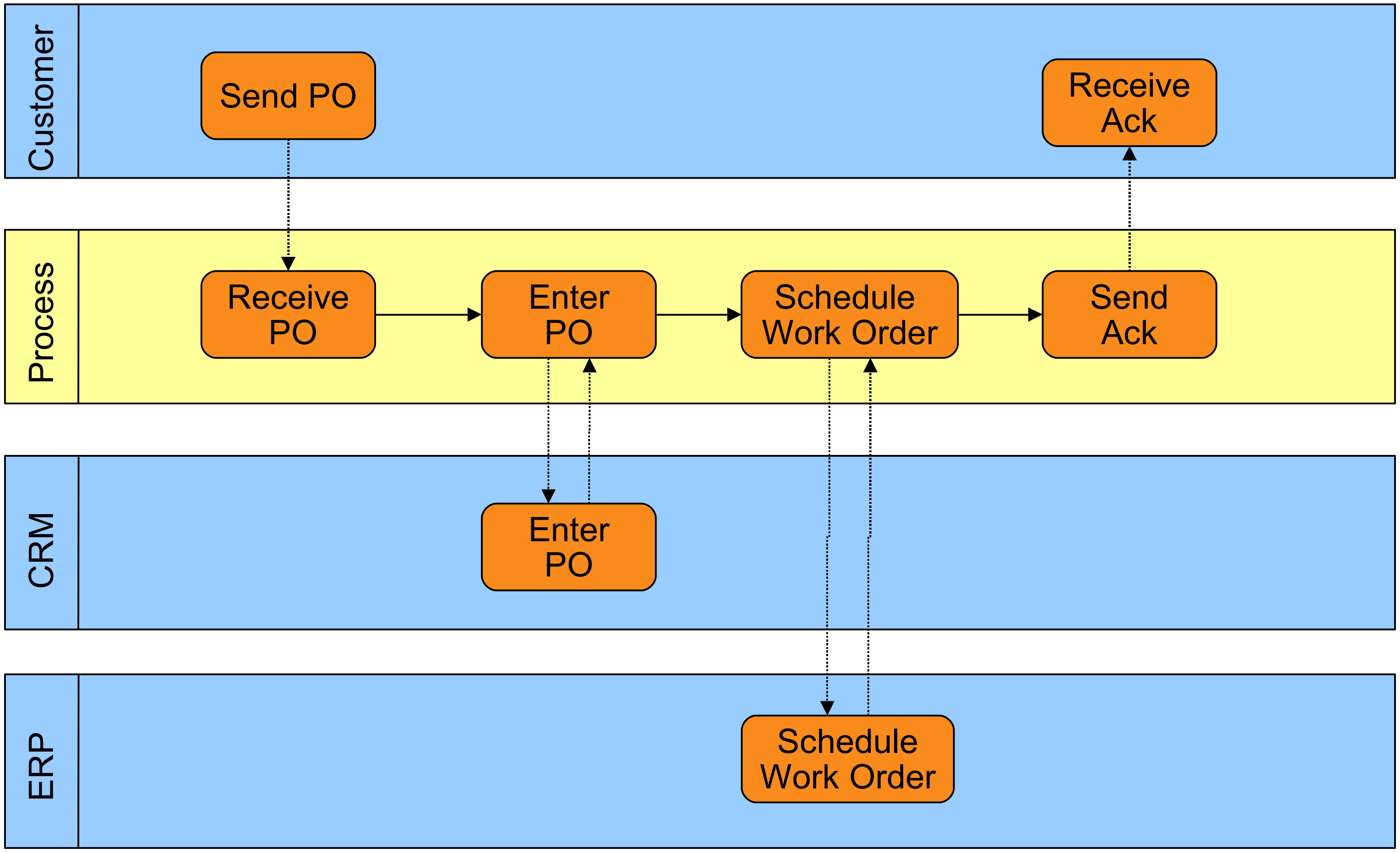
* and because XML was still “hip” back in 2003

```
<process name="HelloWorld">
  <sequence>
    <receive partnerLink="User"
             portType="HelloInterface"
             operation="sayHello"
             variable="helloRequest"
             createInstance="true" />

    <assign>
      <from>concat('Hello ', $helloRequest.text)</from>
      <to>$helloResponse.text</to>
    </assign>

    <reply partnerLink="User"
           portType="HelloInterface"
           operation="sayHello"
           variable="helloResponse" />

  </sequence>
</process>
```

Simple Use-Case

- Customer sends purchase order (PO)
- Company needs to:
  - Enter the purchase order in Customer Relationship Management (CRM) application
  - Create a work order in the Enterprise Resource Planning (ERP) application
  - Send back acknowledgement to customer
- Want to gracefully handle all kinds of errors

Process Diagram

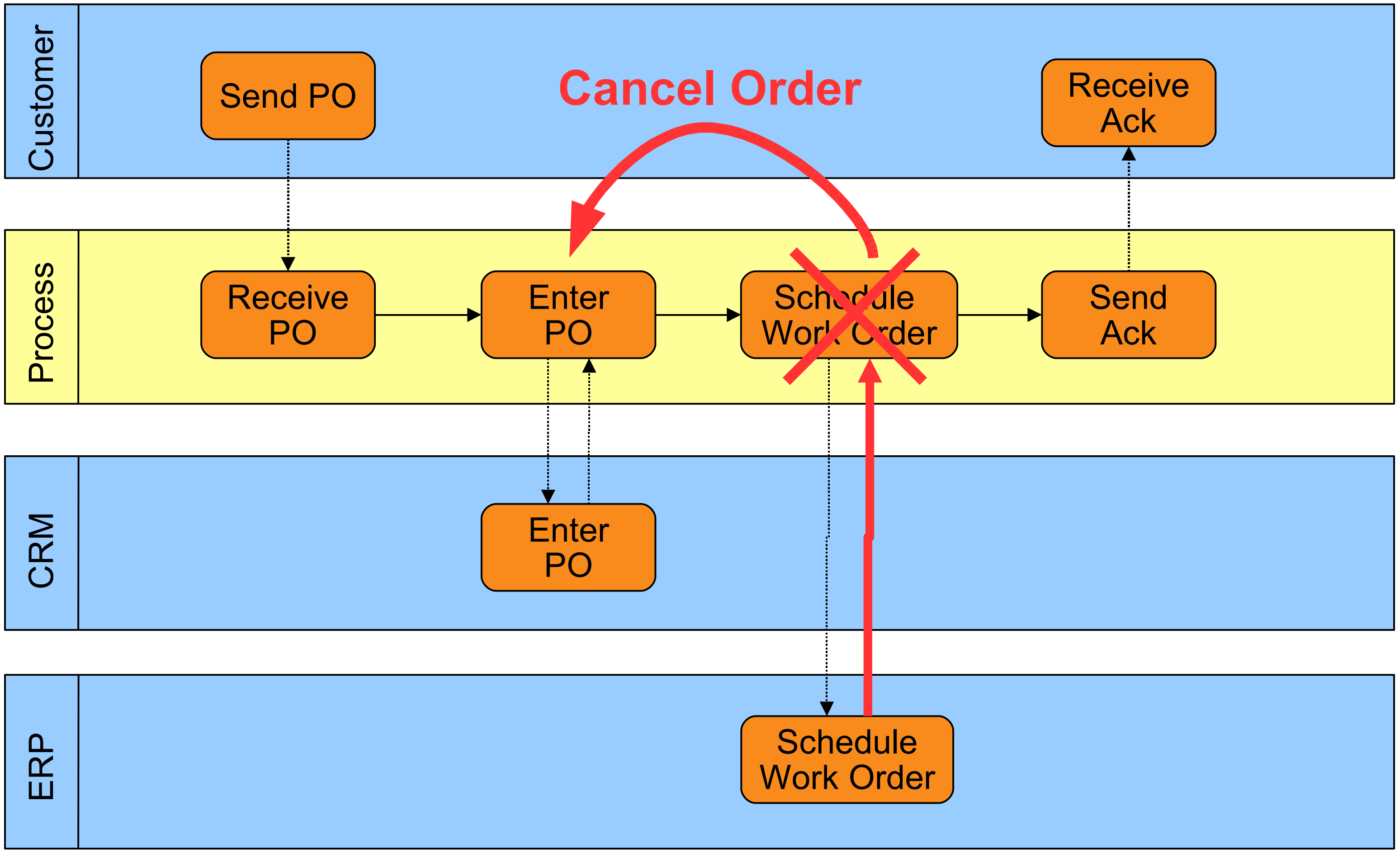


Business Process Modeling Notation (BPMN)

BPEL

```
<process>
  <sequence>
    <receive name="ReceivePO"
             partnerLink="Customer" .../>
    <invoke  name="EnterPO"
             partnerLink="CRM" .../>
    <invoke  name="ScheduleWO"
             partnerLink="ERP" .../>
    <invoke  name="SendAck"
             partnerLink="Customer" .../>
  </sequence>
</process>
```

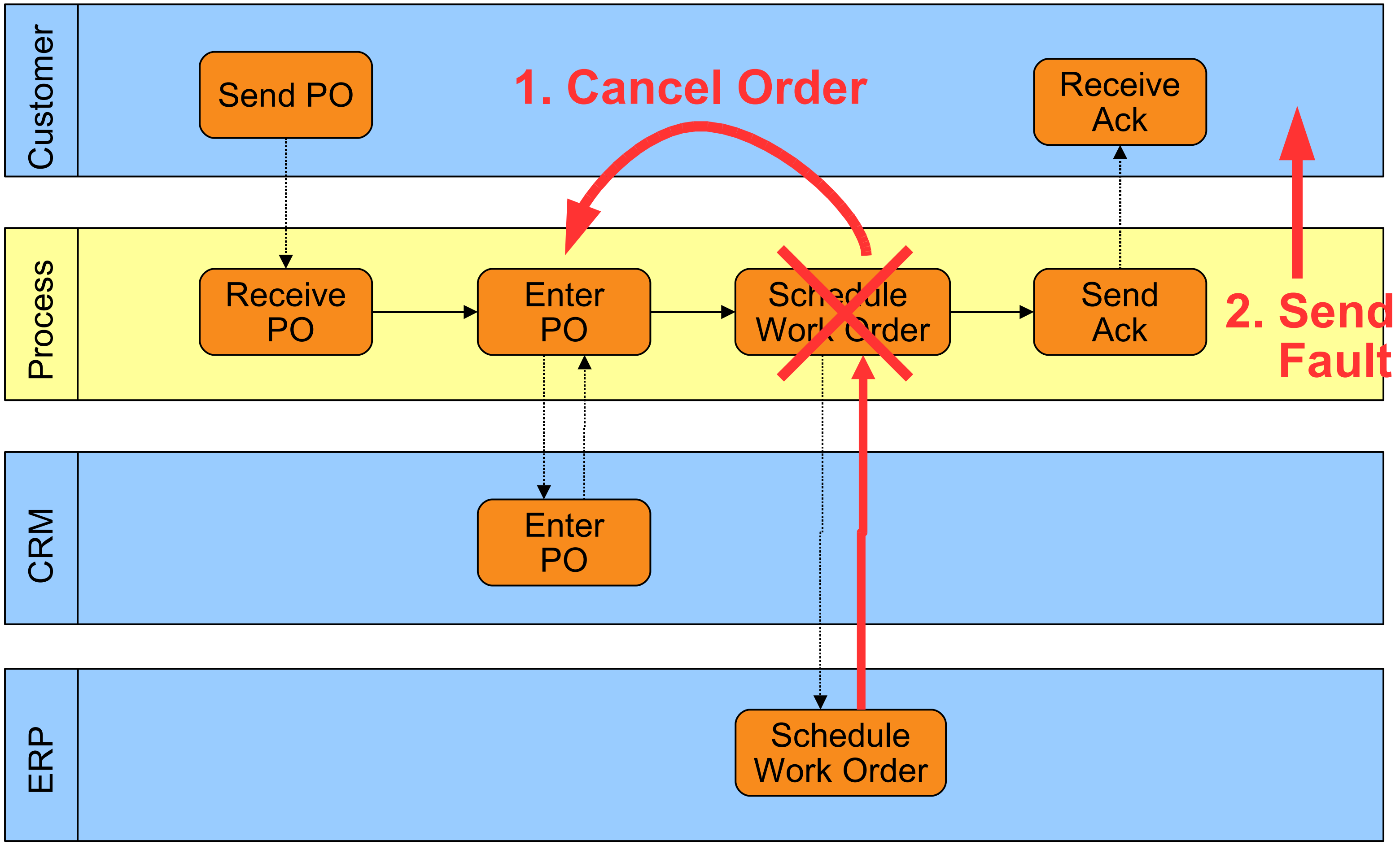
Compensation



Add compensation

```
<process>
  <sequence>
    ...
    <invoke name="EnterPO"
            partnerLink="CRM" ...>
      <compensationHandler>
        <invoke name="CancelPO"
                partnerLink="CRM" .../>
      </compensationHandler>
    </invoke>
    ...
  </sequence>
</process>
```

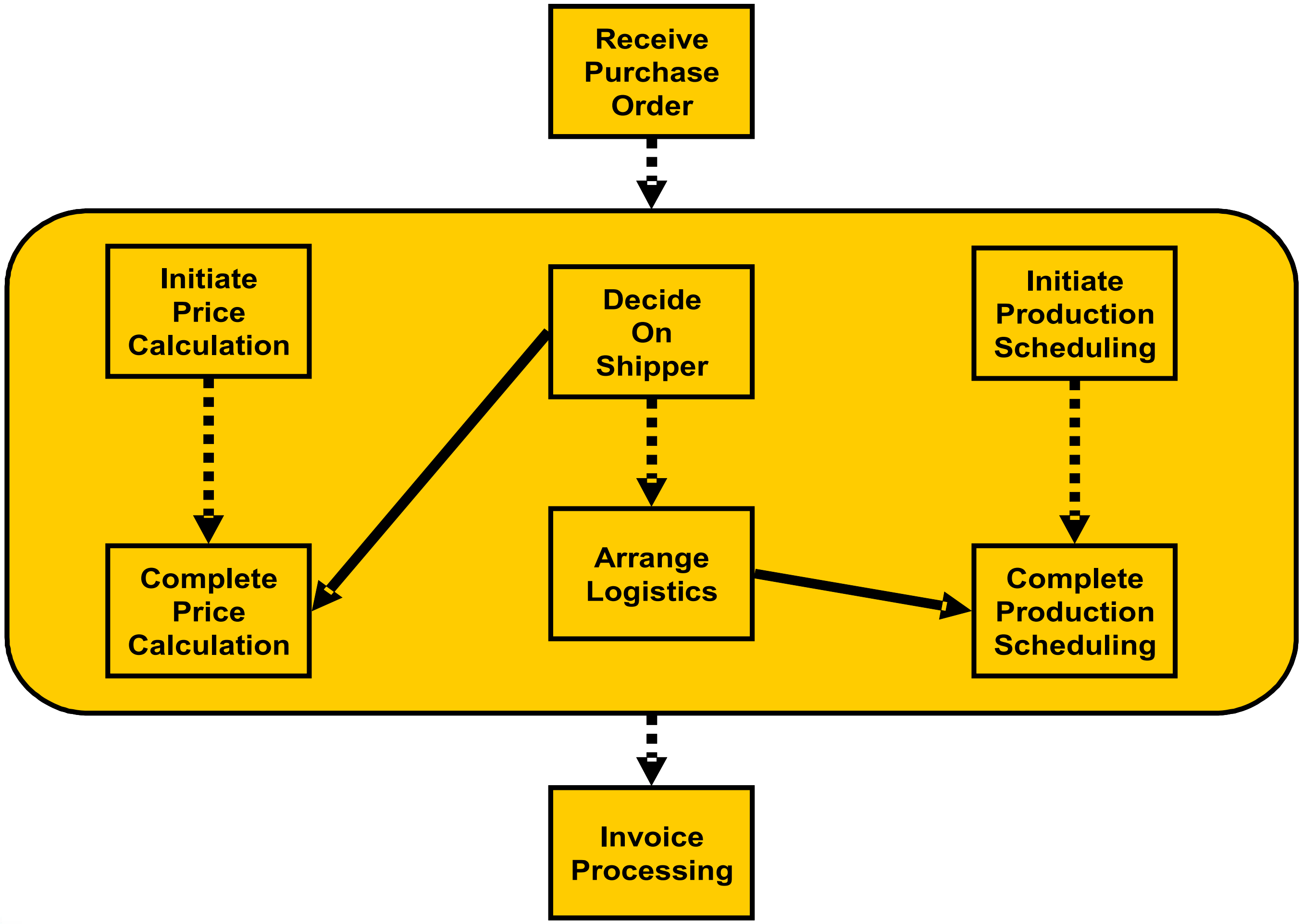
Compensate and Send Fault



Add fault handler

```
<process>
  <sequence>
    ...
    <faultHandler>
        <catchAll>
          <compensate/>
          <invoke name=”SendFault”
                  partnerLink=”Customer”
                  variable=”Fault” .../>
          <rethrow/>
        </catchAll>
    </faultHandler>
  </sequence>
</process>
```

Flow Example


Receive Purchase Order
Initiate Price Calculation
Decide On Shipper
Initiate Production Scheduling
Complete Price Calculation
Arrange Logistics
Complete Production Scheduling
Invoice Processing

Apache Ode

- Great Software
  - A high-performance process engine supporting the BPEL 1.1 and 2.0 specifications
  - Many innovative features
- Cool People
  - A rich and diverse community of users and developers that are participating in SOA projects worldwide

Project History

- March 2006
  - Started in incubator
  - Code donations from Intalio and Sybase
  - Merge with Apache Agila
- July 2007
  - Graduated as top-level project (TLP)
- August 2007
  - Release 1.1: Bug fixes, performance, BPEL
- January 2008
  - Release 1.1.1: Bug fixes

Project Statistics

- Today
  - 10 committers
  - 150 emails/month (ode-user and ode-dev combined)
  - 300 unique visitors per day

BPEL Compliance

- Support for both BPEL 1.1 and 2.0
  - Interoperable with existing tools
  - Many migration success stories
  - Details of compliance can be found on web site

Deployment Architectures

- Webapp/Servlet (Axis2)
  - Deploy as a .war on any servlet container (e.g. Apache Tomcat)
- Java Business Integration (JBI)
  - Deploy as a service engine on JBI 1.0 container (e.g. Apache ServiceMix)
- Scalable Component Architecture (SCA)
  - Deploy on Apache Tuscany (experimental)

Build Your Own

- Embed Apache Ode into your application(s)
- Pluggable Dependencies
  - Scheduler
  - Transaction manager
  - DataSource / Data Access Object(s)
  - Message bus
  - Deployment strategy
  - Event listeners, ...

Engine Features

- Robustness
  - Automatic process suspension upon transient failure, automatic retries, etc.
  - Everything executed within transaction (XA) internally; option to detach
- Deployment
  - Hot-deployment
  - Process versioning

Engine Features

- Implicit message correlation
  - Based on stateful protocol exchange (WS-Addressing or HTTP)
- Protocols
  - SOAP/HTTP
  - Plain-Old XML (HTTP binding)
  - JMS
  - ...

Scalability

- Process definitions
  - Activation/passivation of process definitions
  - Tested 100,000+ definitions on single JVM
- Process instances
  - In-Memory: Limited only by available RAM
  - Persistent: Limited by database storage

Performance

- Varies depending on choices
  - Message bus (Axis2 / JBI / SCA / ...)
  - Process design
  - Process persistence
  - Event persistence
  - Hardware/software configuration
- My experience: Database is always the bottleneck
  - Invest in a good disk subsystem and tune your database server
  - Partition services over multiple server instances

Management Features

- Process Management API
  - Rich querying
  - Suspend, resume instances
  - Deploy, undeploy, activate, retire
- Debugging API
  - Step through process execution
  - Watch variables, etc.
- Configurable event/audit trail
  - Record to file, database, JMS, ...

BPEL Extensions (1)

- XPath 2.0 and XSLT 2.0
  - More expressive assignments, expressions and transformations
- Atomic Scopes
  - All-or-nothing units of work; mapped unto XA transactions
- JavaScript E4X
  - More concise assignments/expressions

BPEL Extensions (2)

- External variables
  - Data lives beyond instance lifespan
  - Map to database table row
  - Map to REST resource (not available yet)
- Read/write message headers
  - SOAP, HTTP, ...
- XPath extension functions
  - It's easy to write your own

Best Practices (1)

- Use tooling!
  - Eclipse BPEL Designer
  - NetBeans BPEL Designer
  - SOAPUI
  - Eclipse STP BPMN Modeler
  - Many commercial offerings
- Get trained
  - XML, XSD, XPath, XSLT, WSDL, WS-*, BPEL, BPMN, ...

Best Practices (2)

- Web-Services
  - Coarse-grained services
  - Follow standards (WS-I BasicProfile)
  - Everything you already know about WS
- Use BPEL 2.0 standard
  - More interoperable
  - Better defined semantics

Best Practices (3)

- Process lifespan
  - Use process composition to control instance lifespan
  - Typically a fraction of the rate of change
- Avoid modeling entities as processes
  - An order is not a process

Best Practices (4)

- Abstract out business rules
  - Use XPath function extensions or WS
  - Consider rule engine for complex or large number of rules
- Keep processes protocol-independent
  - Avoid SOAP header manipulation
  - Avoid authentication/authorization in process

What's Coming

- Simple BPEL (SimPEL)
  - Programmer-friendly syntax similar to JavaScript
  - More dynamic typing
- RESTful BPEL
  - Invoke REST services natively
  - Expose process a set of resources
- Event multicasting
  - Signaling between processes
- Event feeds
  - Scalable event listening model based on Atom

What's Coming

- Administration console (Google SoC)
  - Manage processes, instances, querying, ...
- Human workflow (Singleshot)
  - BPEL4People extensions
  - Workflow services
  - Task list webapp
- Better support for WS-* standards
  - WS-Security, WS-ReliableMessaging, ...

HelloWorld, Revisited

```
# SimPEL Syntax
process HelloWorld {
  partnerLink user;
  variable request, response;

  request = receive(user, sayHello);
  response.text = ”Hello ” + request.text;
  reply(user, sayHello, response)
}
```

```
# Loan approval process (BPEL 2.0 Spec; Section 15.3)
process LoanApproval {
  partnerLink customer, assessor, approver;
  try {
    parallel {
      request = receive(customer, request);
      signal(receive-to-assess, [$request.amount < 10000]);
      signal(receive-to-approval, [$request.amount >= 10000]);
    } and {
      join(receive-to-assess);
      risk = invoke(assessor, check);
      signal(assess-to-setMessage, [$risk.level = 'low']);
      signal(assess-to-approval, [$risk.level != 'low']);
    } and {
      join(assess-to-setMessage);
      approval.accept = "yes";
      signal(setMessage-to-reply);
    } and {
      join(receive-to-approval, assess-to-approval);
      invoke(approver, approve);
      signal(approval-to-reply);
    } and {
      join(approval-to-reply, setMessage-to-reply);
      reply(customer, request, approval);
    }
  } catch(loanProcessFault) { |error|
    reply(customer, request, error);
  }
}
```

Remember One Thing

- Greenspun's Tenth Rule of Programming
  - Any sufficiently complicated program contains an ad-hoc, informally-specified, bug-ridden, slow implementation of half of ~~**Lisp**~~ **BPEL**
  - Don't write your own business process engine

Join us! :)

ApacheCon

Questions?

Thank You!