The BPM dream

A slide show based on two papers

Mapping BPMN to BPEL
http://eprints.qut.edu.au/5266/1/5266.pdf
Ouyiang, Dumas, van der Aalst and ter Hofstede

The Seven Fallacies of Business Process Execution
http://www.infoq.com/articles/seven-fallacies-of-bpm
Jean-Jacques Dubray

Business Process Management (BPM)

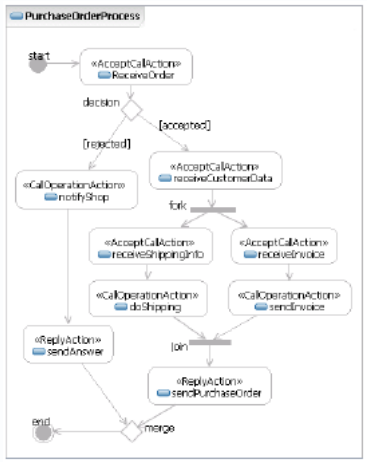
- BPM is a discipline for building, maintaining and evolving enterprise systems on the basis of business process models.

- A business process model or diagram is a flow chart-style representation of activities leading from a start event to result or goal, e.g.
  - processing a customer request or complaint,
  - satisfying a regulatory requirement
  - etc.

The Business Process Management dream

- Given
  - Business Process Modelling Notation (**BPMN**)
  - Business Process Execution Language **(BPEL)**

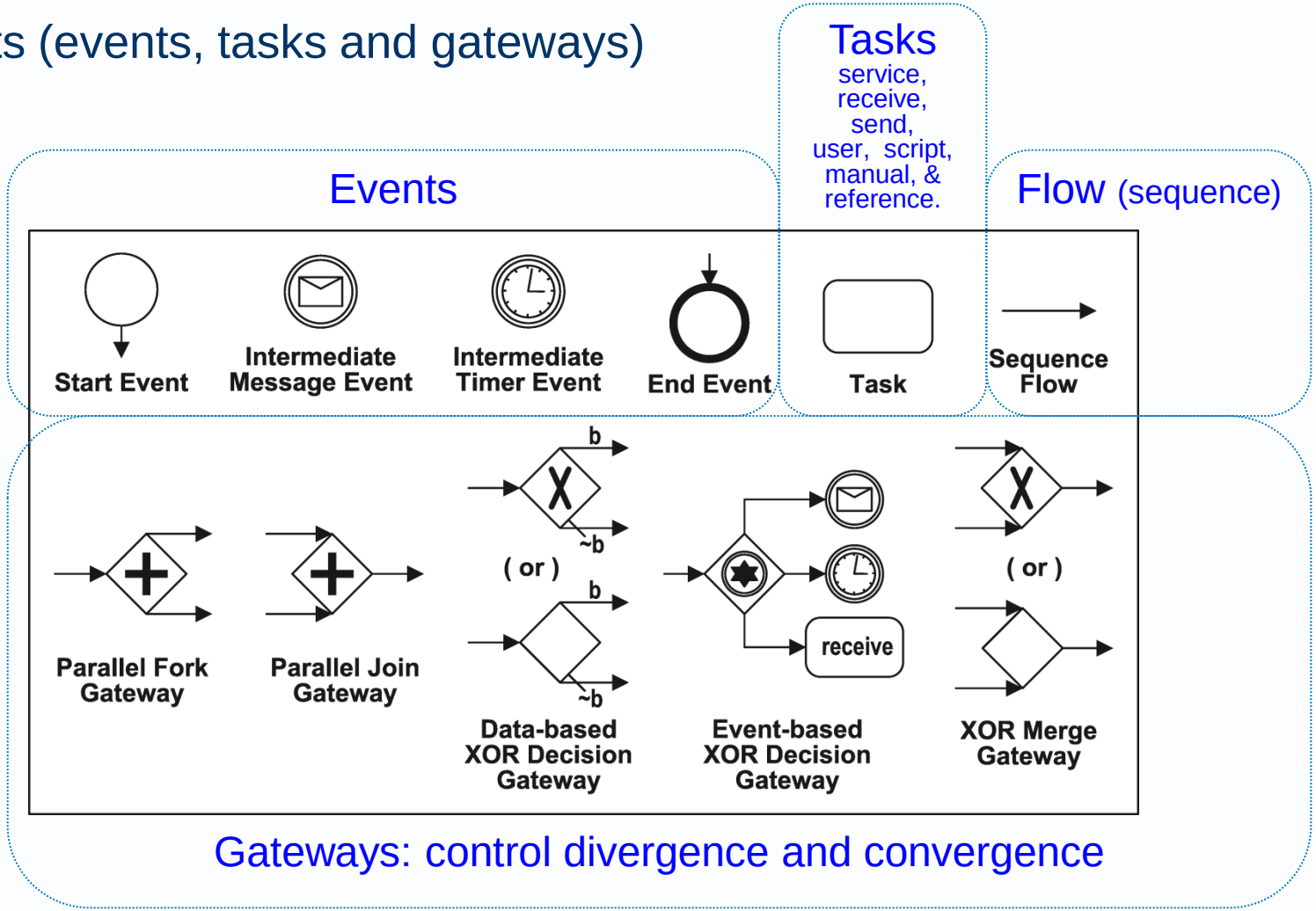
- The dream is
  - analysts use **BPMN** to visualize business processes and
  - developers transform the visualizations to **BPEL** for execution.




```
<process name="PurchaseOrderProcess">
<sequence>
  <receive name="ReceiveOrder"/>
  <switch>
   <case condition="[accepted]">
     <sequence>
       <receive name="receiveCustomerData"/>
       <flow>
         <sequence>
           <receive name="receiveShippingInfo"/>
           <invoke name="doShipping"/>
         </sequence>
         <sequence>
           <receive name="receiveInvoice"/>
           <invoke name="sendInvoice"/>
         </sequence>
       </flow>
       <reply name="sendPurchaseOrder"/>
     </sequence>
   </case>
   <case condition="[rejected]">
     <sequence>
       <invoke name="notifyShop"/>
       <reply name="sendAnswer"/>
     </sequence>
   </case>
  </switch>
</sequence>
</process>
```

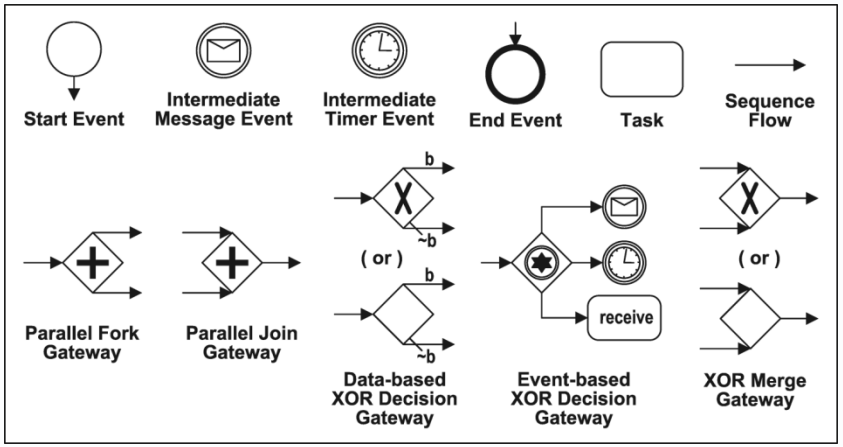
BPMN core elements only

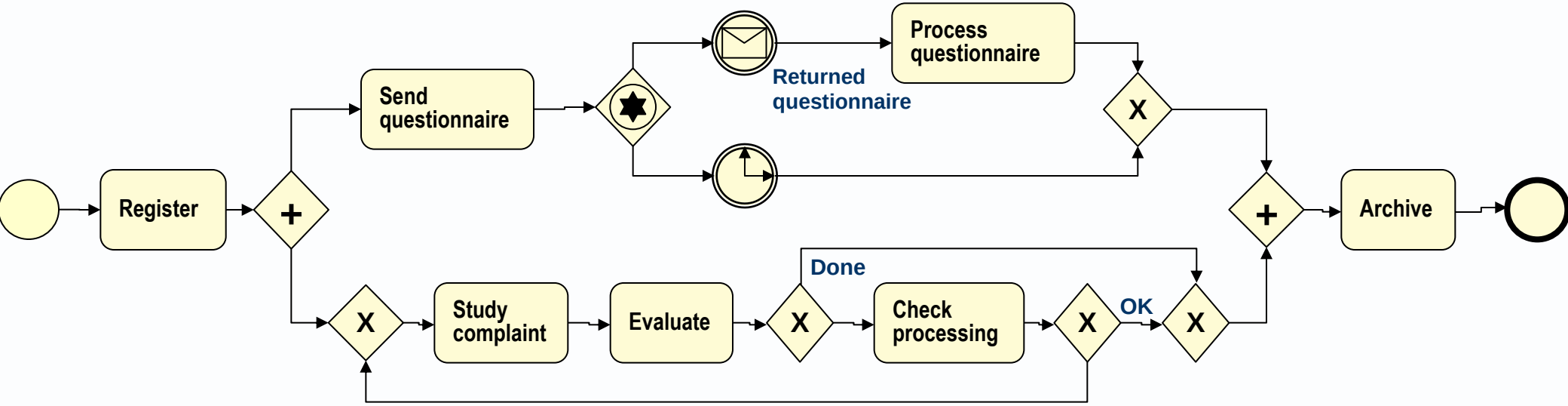
- Objects (events, tasks and gateways)
- Flows



Business Process Diagram (BPD)

- A BPD is a flowchart
- Made of BPMN elements.



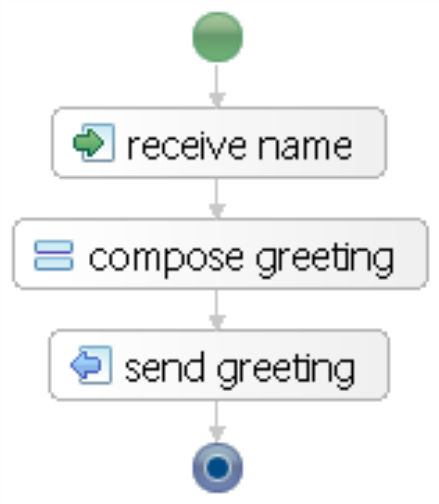


Business Process Execution Language for Web Services (BPEL)

- BPEL is an XML-based language that allows Web Services to interconnect and share data..

- Basic activities = atomic actions such as:
  - **invoke,** invoking an operation on a Web service;
  - **receive,** waiting for a message from a partner;
  - **exit,** terminating the entire Web Service instance;
  - **empty,** doing nothing
- Structured activities connect basic activities:
  - **sequence,** for defining an execution order;
  - **flow,** for parallel routing;
  - **switch,** for conditional routing;
  - **pick,** for race conditions based on timing or external triggers;
  - **while,** for structured looping; and
  - **scope,** for grouping activities into blocks to which event, fault and compensation handlers may be attached.

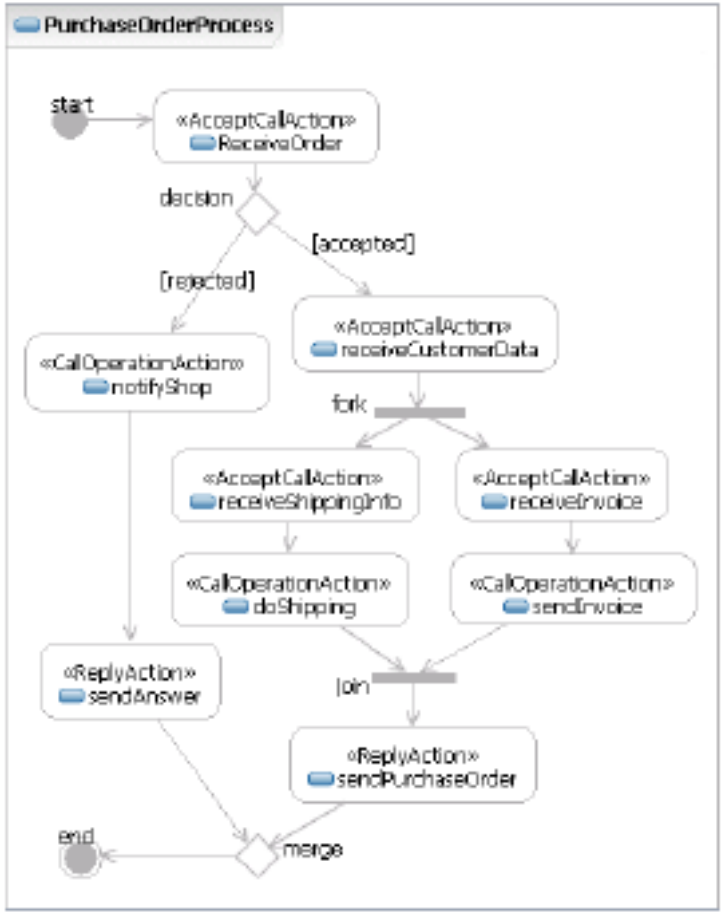
Hello world example




```
<process name="HelloWorld" targetNamespace="http://jbpm.org/examples/hello"
  xmlns:tns="http://jbpm.org/examples/hello"
  xmlns:bpel="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">

  <partnerLinks>
    <!-- establishes the relationship with the caller agent -->
    <partnerLink name="caller" partnerLinkType="tns:Greeter-Caller" myRole="Greeter" />
  </partnerLinks>

  <variables>
    <!-- holds the incoming message -->
    <variable name="request" messageType="tns:nameMessage" />
    <!-- holds the outgoing message -->
    <variable name="response" messageType="tns:greetingMessage" />
  </variables>

  <sequence name="MainSeq">

    <!-- receive the name of a person -->
    <receive name="ReceiveName" operation="sayHello" partnerLink="caller"
      portType="tns:Greeter" variable="request" createInstance="yes" />

    <!-- compose a greeting phrase -->
    <assign name="ComposeGreeting">
      <copy>
        <from expression="concat('Hello, ', bpel:getVariableData('request', 'name'), '!')" />
        <to variable="response" part="greeting" />
      </copy>
    </assign>

    <!-- send greeting back to caller -->
    <reply name="SendGreeting" operation="sayHello" partnerLink="caller"
      portType="tns:Greeter" variable="response" />

  </sequence>

</process>
```

http://www.jot.fm/issues/issue_2007_10/paper13/images/figure11.gif

UML Activity Diagram



BPEL description.

```
<process name="PurchaseOrderProcess">
<sequence>
  <receive name="ReceiveOrder"/>
  <switch>
   <case condition="[accepted]">
     <sequence>
       <receive name="receiveCustomerData"/>
       <flow>
         <sequence>
           <receive name="receiveShippingInfo"/>
           <invoke name="doShipping"/>
         </sequence>
         <sequence>
           <receive name="receiveInvoice"/>
           <invoke name="sendInvoice"/>
         </sequence>
       </flow>
       <reply name="sendPurchaseOrder"/>
     </sequence>
   </case>
   <case condition="[rejected]">
     <sequence>
       <invoke name="notifyShop"/>
       <reply name="sendAnswer"/>
     </sequence>
   </case>
  </switch>
</sequence>
</process>
```

Mapping BPMN to BPEL

http://eprints.qut.edu.au/5266/1/5266.pdf

Ouyiang, Dumas, van der Aalst and ter Hofstede

A slide show based on the paper above

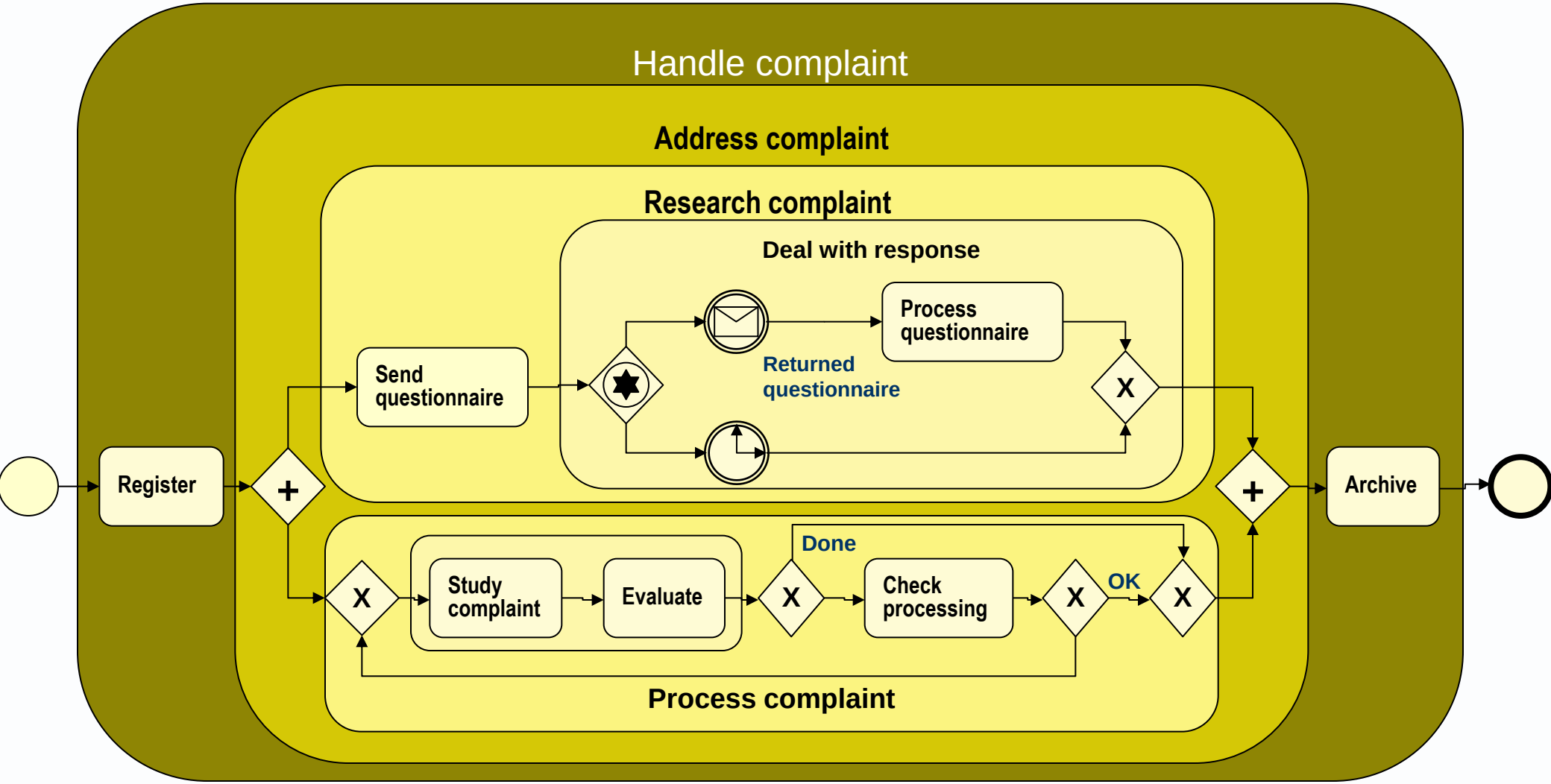
A process for mapping BP diagrams to BPEL

"To map a BPD onto (readable) BPEL code,
we need to transform a graph structure into a block structure.

For this purpose, we [de]compose a BPD into components.
A component is a subset of the BPD that has one entry and one exit point.
We then try to map components onto suitable BPEL blocks".

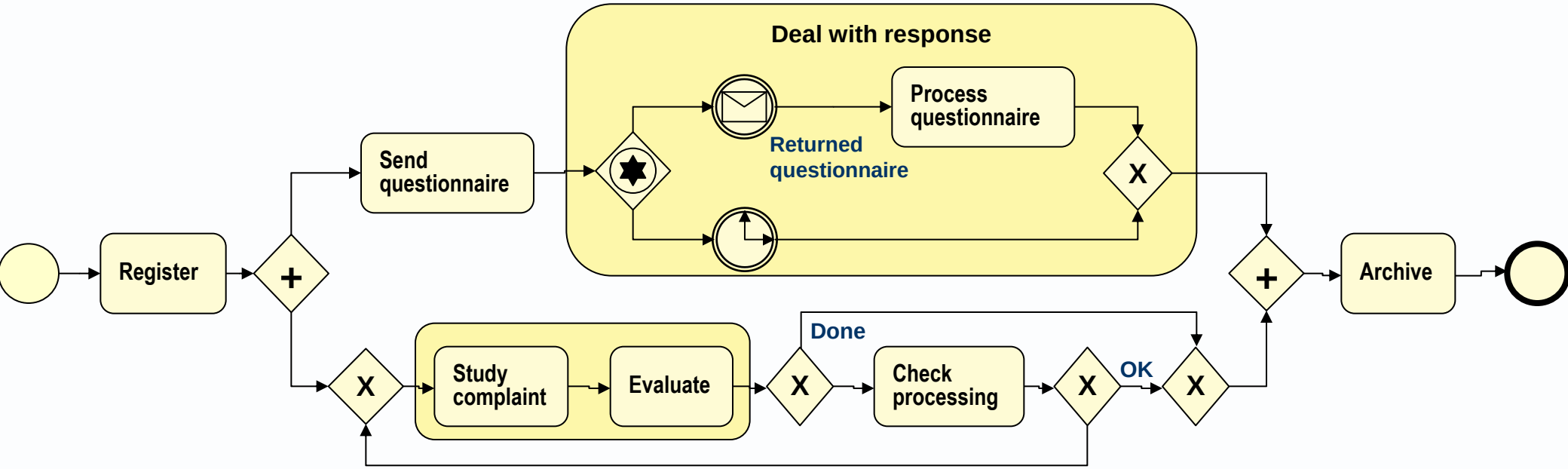
http://eprints.qut.edu.au/5266/1/5266.pdf
Ouyiang, Dumas, van der Aalst and ter Hofstede

Process composition / decomposition example



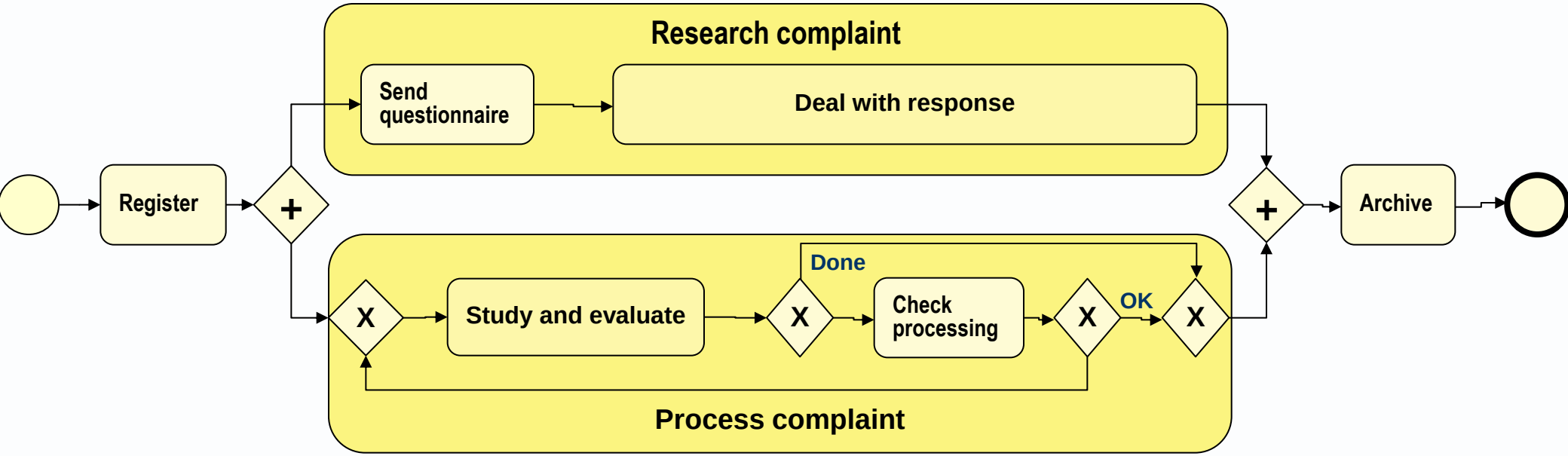
Complaint handling – 1st level composition

- Create components with one entry and exit point
  - E.g. Deal with response



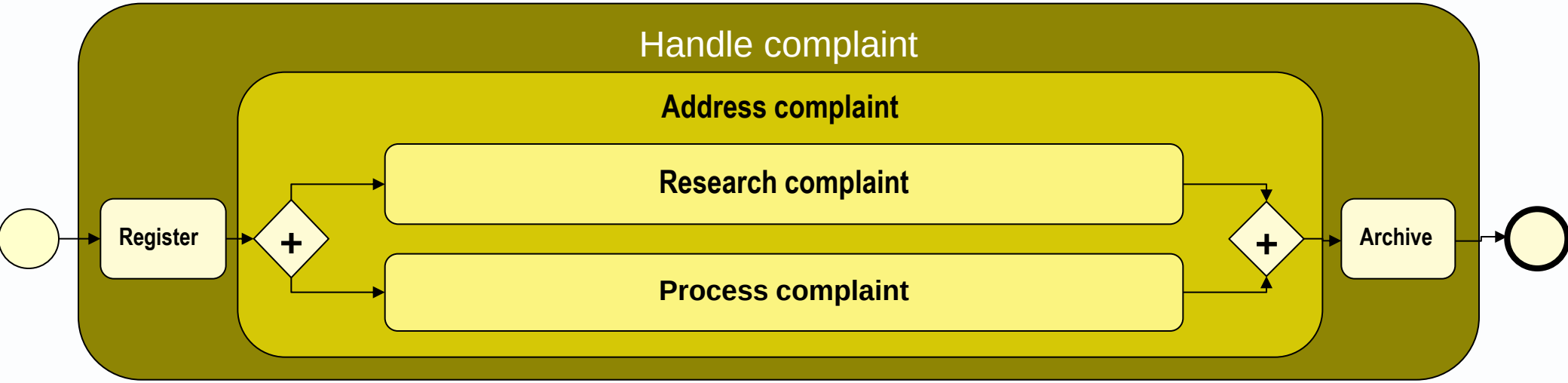
Complaint handling – 2nd level composition

- Higher-level components with one entry and exit point
  - "Research complaint" is a plain "sequence" in BPEL
  - "Process complaint" (being cyclic) is a more complex "scope" in BPEL



Complaint handling - 3rd and 4th level composition

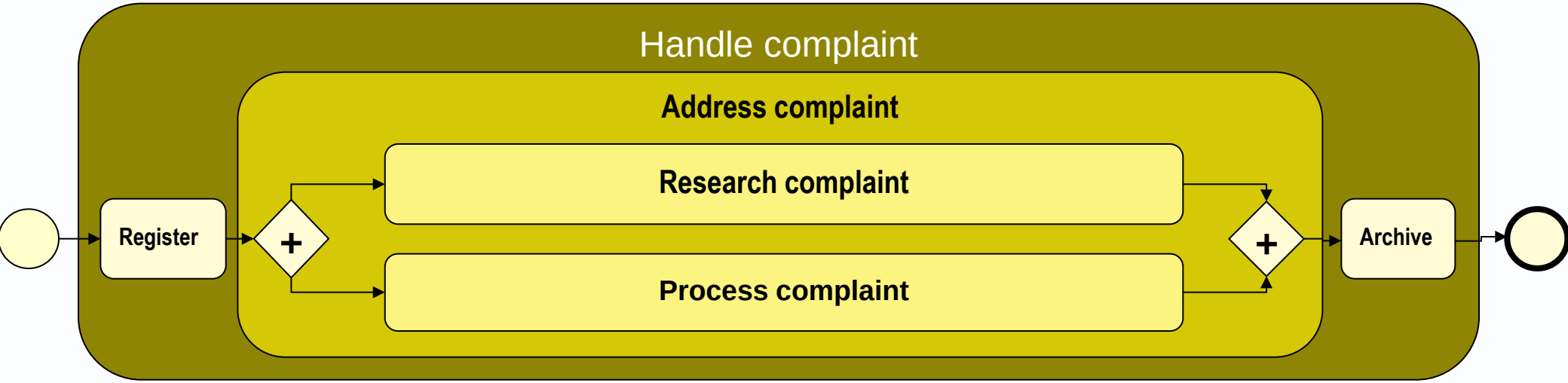
- "Handle complaint" is a plain "sequence" in BPEL
  - "Address complaint" is composed of two parallel components in BPEL
    - "Research complaint" is a plain "sequence" in BPEL
    - "Process complaint" (being cyclic) is a more complex "scope" in BPEL



A BPD represented as a hierarchical structure of BPEL

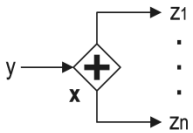
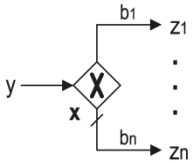
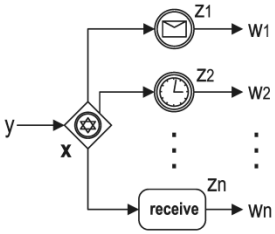
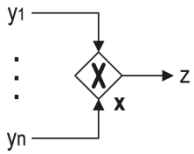
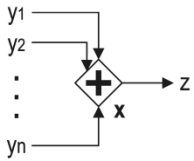
```
<process name="complaint handling">
    <sequence name="Handle complaint ">
        <invoke name="register">
        <flow name="Address complaint "> ... </flow>
        <invoke name="archive">
    </sequence>
</process>
```

```
<flow name=" Address complaint ">
    <sequence name="Research complaint "> ... </sequence>
    <scope name="Process complaint "> ... </scope>
</flow>
```



Mapping BP Diagrams to BPEL

- Given a component of a process, with one entry and exit point
- If it is well-structured, then it can be directly mapped onto BPEL structured activities.
- Else if it is acyclic, it may be mappable to control link-based BPEL code.
- Else, the mapping of the component will rely on BPEL event handlers via the usage of event-action rules >>
- [Read the paper for more]

BPMN Object		BPEL Event Handler
Task		<onEvent $e_{y,x}$> <sequence> *Mapping(x)* <invoke $e_{x,z}$/> </sequence> </onEvent>
Event		
Parallel Fork Gateway		<onEvent $e_{y,x}$> <flow> <invoke $e_{x,z1}$/> . . . <invoke $e_{x,zn}$/> </flow> </onEvent>
Data-based XOR Decision Gateway		<onEvent $e_{y,x}$> <switch> <case condition="*b1*"> <invoke $e_{x,z1}$/> <case/> . . . <case condition="*bn*"> <invoke $e_{x,zn}$/> <case/> </switch> </onEvent>
Event-based XOR Decision Gateway		<onEvent $e_{y,x}$> <pick> <onMessage *z1*> <invoke $e_{z1,w1}$/> <onMessage/> <onAlarm *z2*> <invoke $e_{z1,w2}$/> <onAlarm/> . . . <onMessage *zn*> <invoke $e_{zn,wn}$/> <onMessage/> </pick> </onEvent>
XOR Merge Gateway		<onEvent $e_{y1,x}$> <invoke $e_{x,z}$/> </onEvent> . . . <onEvent $e_{yn,x}$> <invoke $e_{x,z}$/> </onEvent>
Parallel Join Gateway		<onEvent $e_{y1,x}$> <sequence> <flow> <receive $e_{y2,x}$/> . . . <receive $e_{yn,x}$/> </flow> <invoke $e_{x,z}$/> <sequence> </onEvent>

Note

- The mapping above uses only some BPMN constructs
- It excludes
  - exception handling
  - OR-joins
  - other advanced constructs

The Seven Fallacies of Business Process Execution

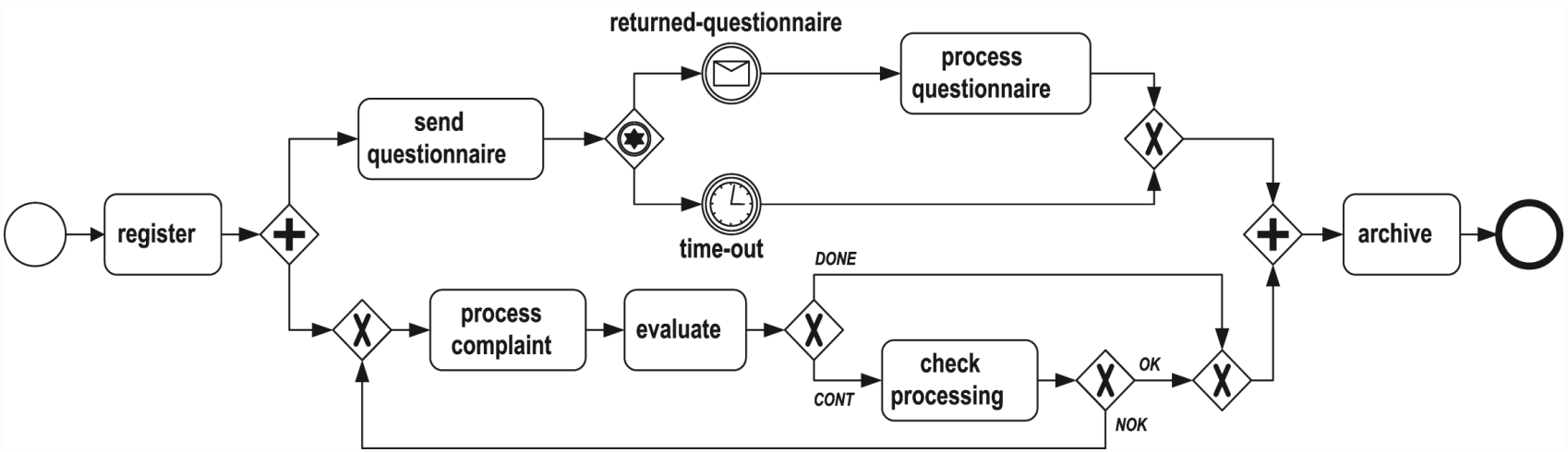
http://www.infoq.com/articles/seven-fallacies-of-bpm

Jean-Jacques Dubray

A slide show based on the paper above

1: Analysts don't model processes from a system's viewpoint

- Analyst's model processes from the *human actor's* viewpoint.
- It usually impossible and unnecessary to prescribe human actions as precisely and completely as computer actions.
- Humans often interpret and adapt the logic and steps of the process

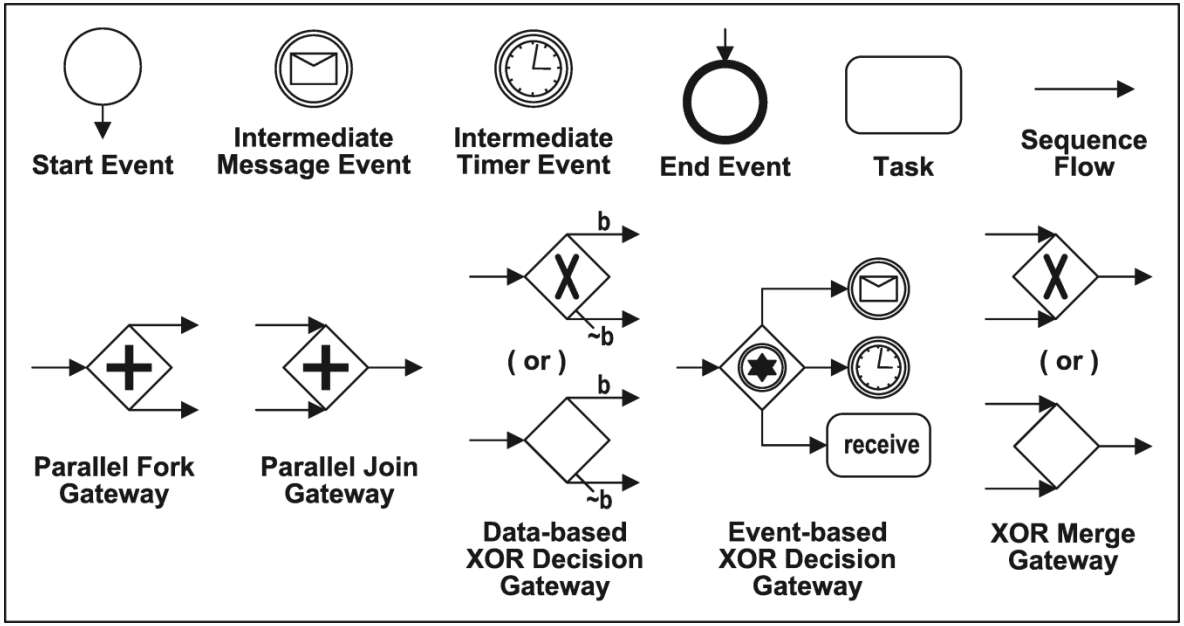


- (Analysts used to be taught to model the life histories of entities in data models more formally, but this is uncommon nowadays.)

2: Business users cannot easily learn BPMN and all its features.

- Business users, analysts, and architects use only small selection of the constructs created by standard writers.

- Not even this much?



- [System modelling languages like BPMN, UML and ArchiMate are abused as much as they are used]

3: Business analysts are unable to create executable solutions from process models

- [This has been a pipe dream since the days of the common business-oriented language (COBOL) in the 1970s.

- The closest we have to system generation tools are 4GLs based on
  - A data model
  - Forms for entering and displaying data
  - Some way of attaching business rules to data elements.

- Where executable solutions are to be produced, agilists may argue that programmers should be doing the business analysis.]

4: There is no silver bullet

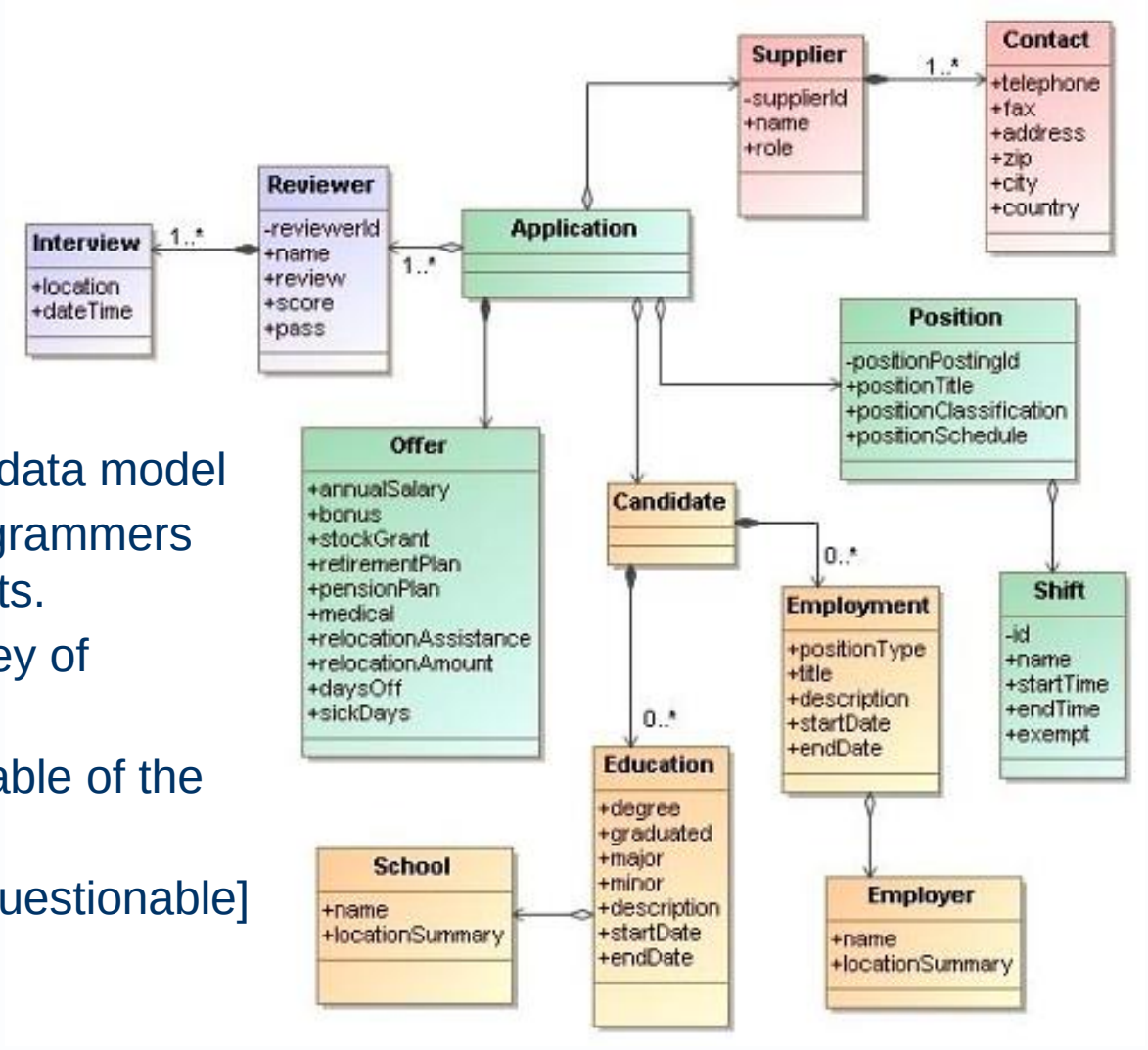
- No magical BPMS can create solutions directly from business analysis diagrams without the need to
  - Design all exception handling (80% of the complexity?)
  - Develop integration with existing systems,
  - Change existing systems of record,
  - Do Quality Assurance.

5: Business Process Execution is not best centralized

- There is only a loose coupling between
  - Less formal Business Process Models (to which BMPN is suited)
  - More formal Entity/Resource Lifecycles (to which BPEL is suited)

- As illustrated by Jean-Jacques Dubray in the following figures
  - Figure 1. The Job Application Data Model
  - Figure 2. The Job Application Lifecycle
  - Figure 3. The Job Application Web Service
  - Figure 4. The Implementation of the Job Application Web Service
  - Figure 5. The Job Application Process

Figure 1. The Job Application Data Model



- [For me, this is an odd data model
- The kind drawn by programmers rather than data analysts.
- Where is the primary key of Application?
- Where is the state variable of the Application life cycle?
- The cardinalities look questionable]

Figure 2. The Job Application Lifecycle

- The business logic of a data entity lifecycle changes rarely
- The business processes that interact with it might change often.

- So, how to implement this Job Application Lifecycle in software?

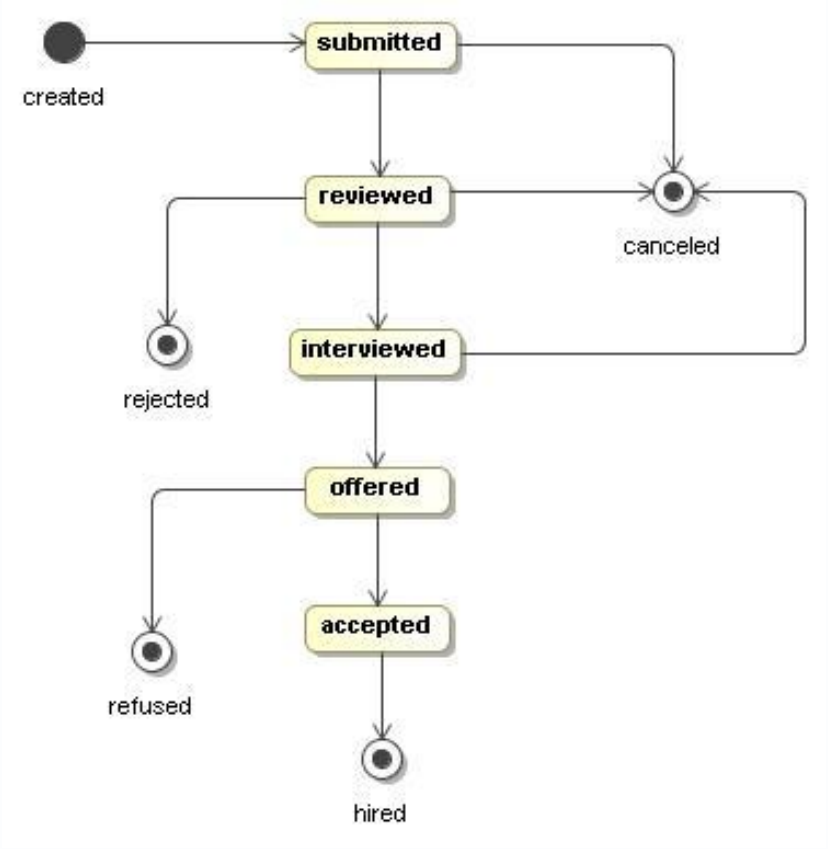


Figure 3. The Job Application Web Service

- This Web Service implements all the actions that result in a state transition in the data entity lifecycle

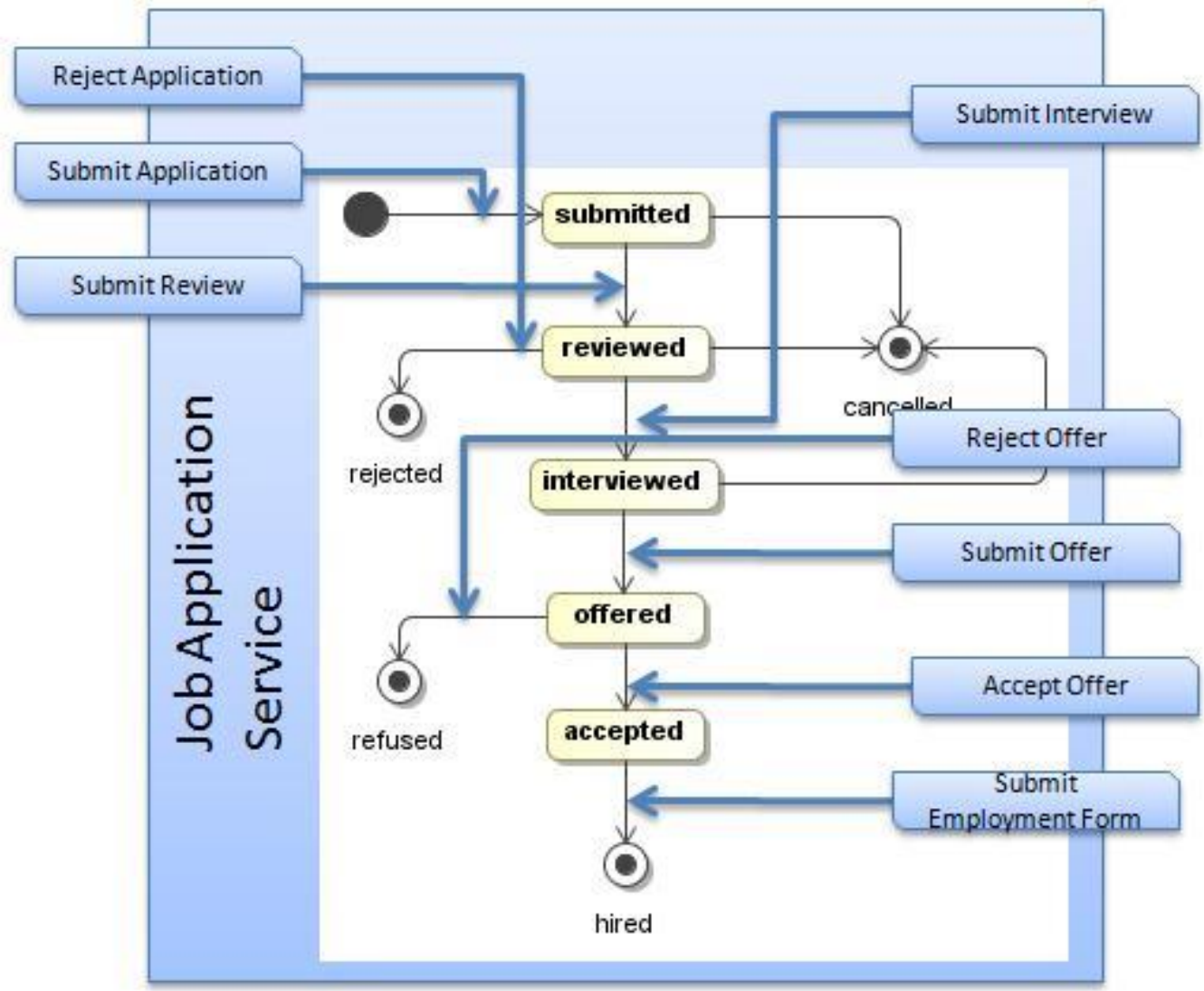
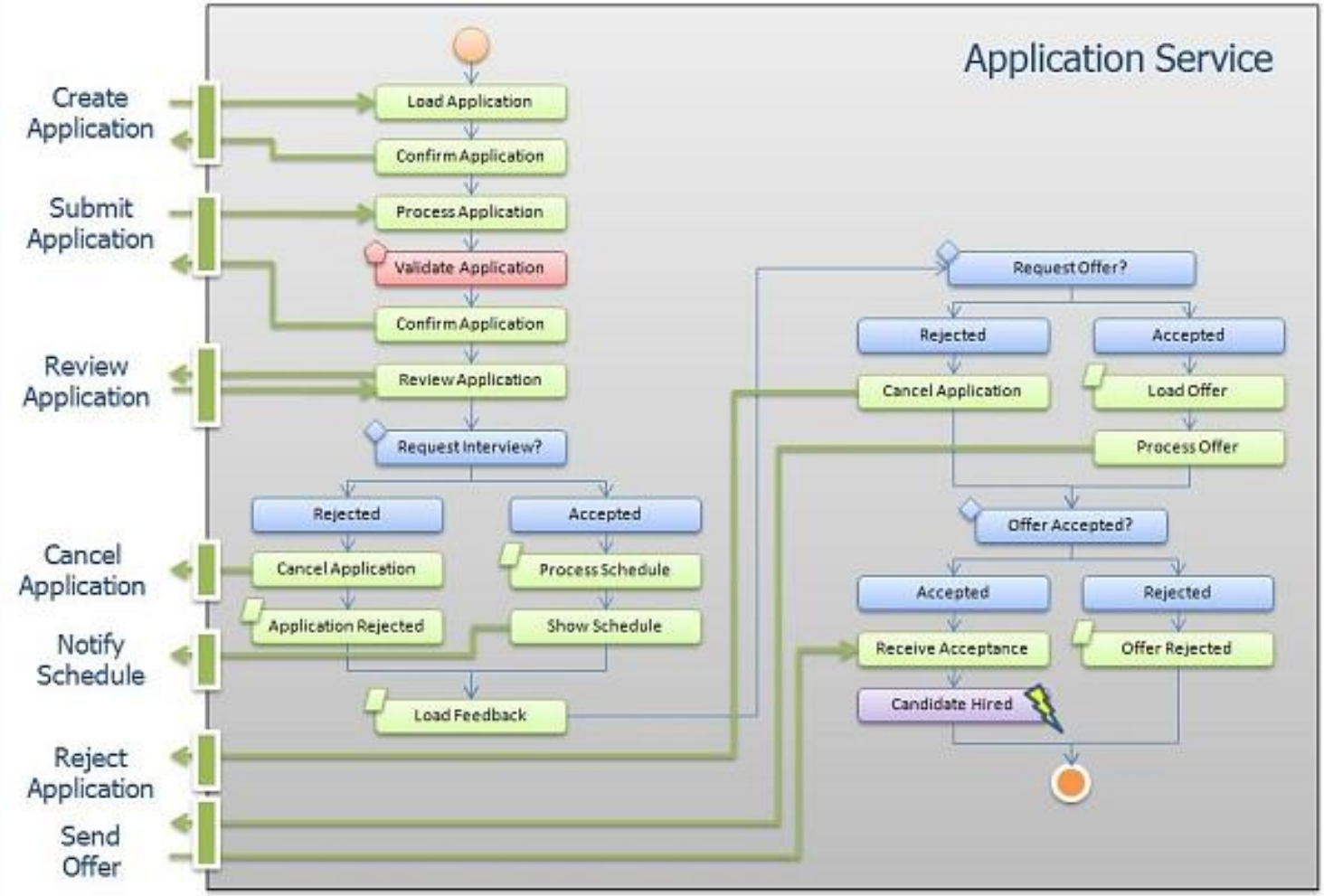


Figure 4. The Implementation of the Job Application Web Service

Avancier

- The BPEL implementation would look like this (using a vendor neutral BPEL notation):



Separating business process from data entity life cycle

"[The] Web Services [implements] the lifecycle of a Job Application independent of processes and activities that may advance the state of the job application.

A process is the set of activities that advance its state.
Resource Lifecycles and processes are decoupled

I don't think anyone can argue with that, yet everyone is trying to model and implement processes without a clear understanding of the resource lifecycles, they are more or less "built-in" the process model."

http://www.infoq.com/articles/seven-fallacies-of-bpm

Figure 5. The Job Application Process

- How a business analyst would create a Job Application business process definition using BPMN
- The groups in Blue represent Human Task boundaries.
- The resource life cycle states have been mapped to process transitions

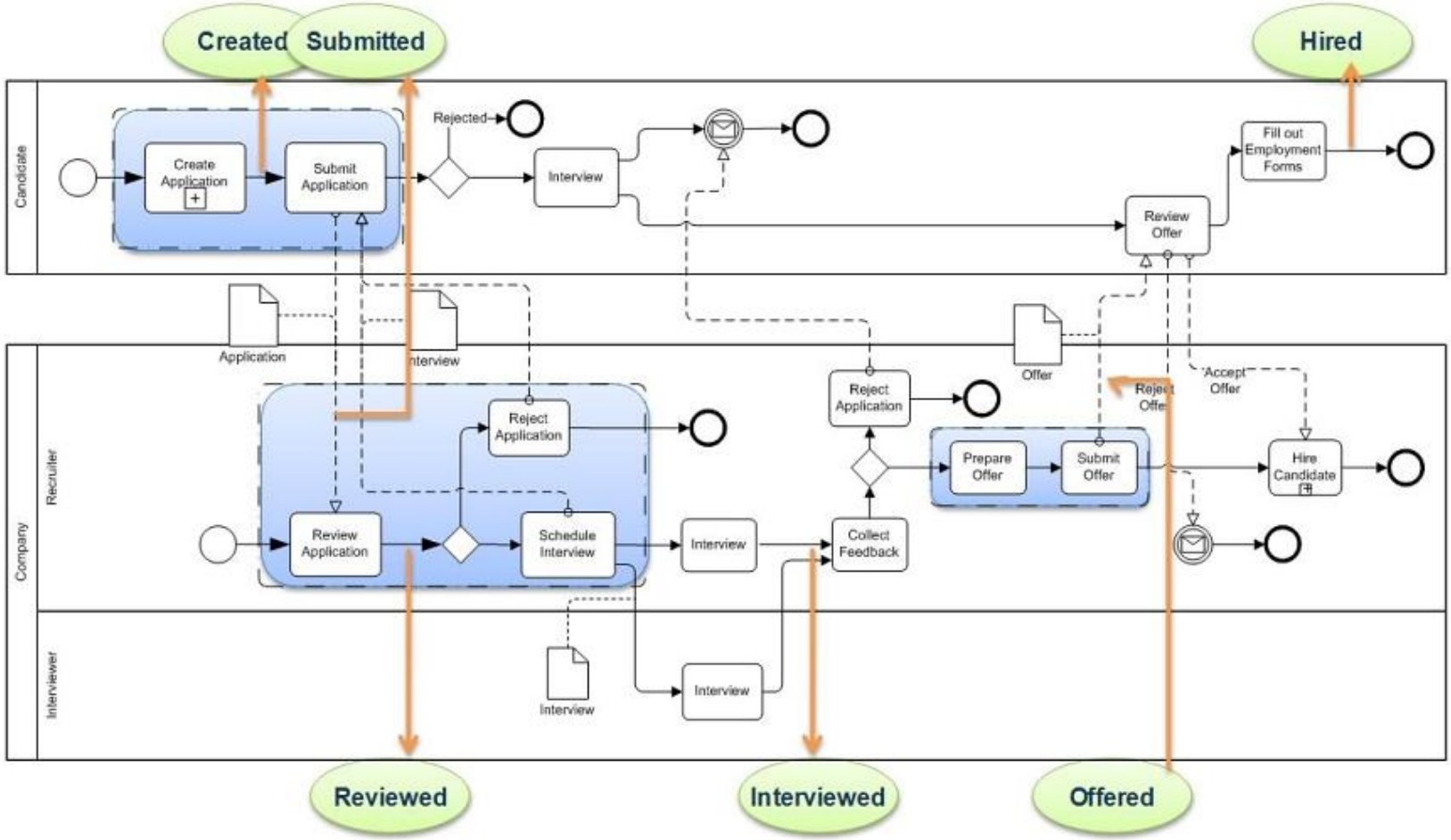
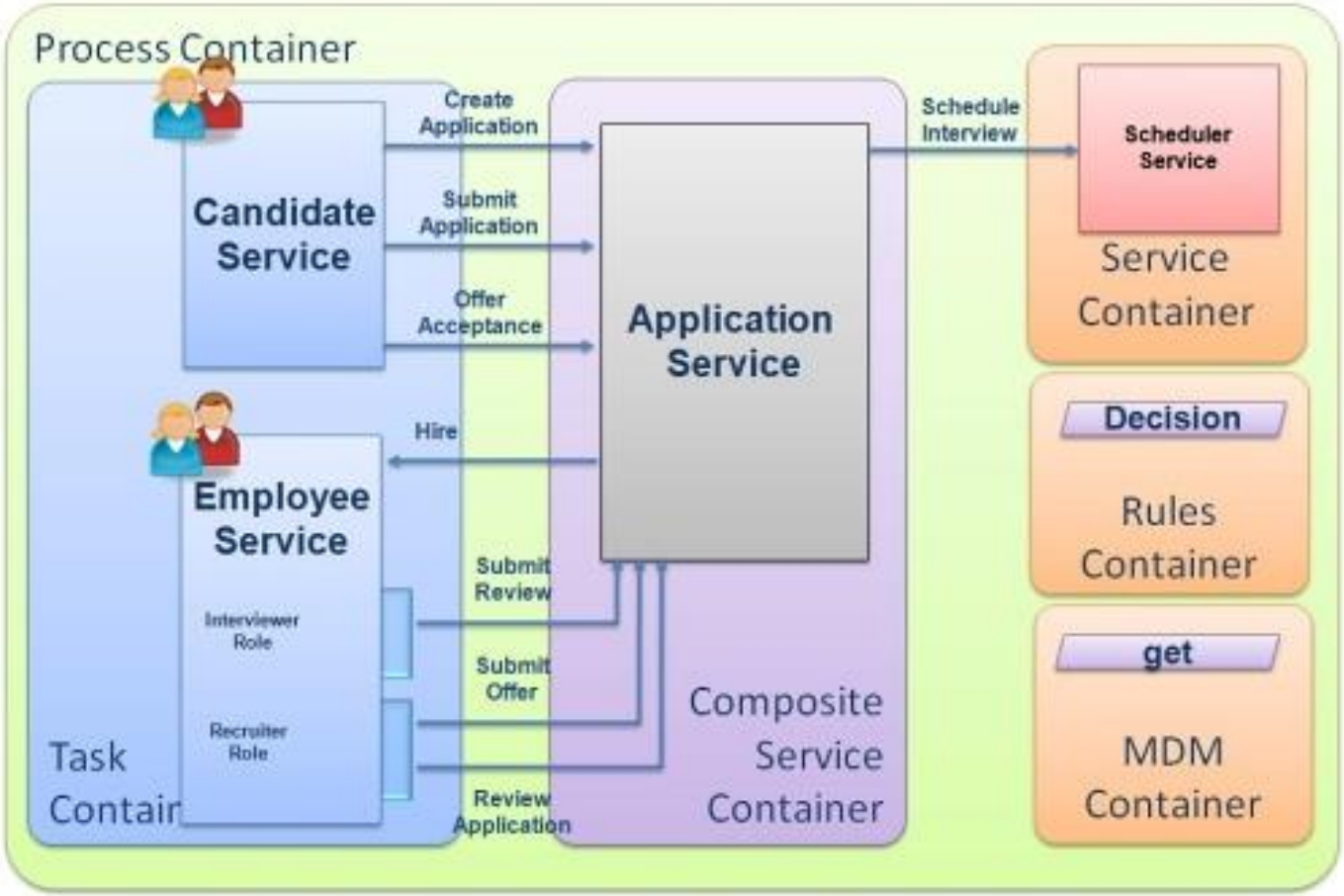


Figure 6. The Job Application Process Implementation

- A business process execution environment is an assembly of Web Services interacting with each other (not a centrally orchestrated set of Web Services)



Seven fallacies of BPM - re-phrased as negatives

1: Business analysts **don't** model processes from a system's viewpoint

2: Business users **cannot** easily learn BPMN and all its features.

3: Business analysts are **unable** to create executable solutions from process models

4: There is no silver bullet

> **No** magical BPMS can create solutions directly from business analysts inputs without the need to develop integrations with existing systems, change existing systems of record and do QA.

5: Business Process Execution is **not** best centralized

6: Business Process Execution semantics **cannot** be derived easily from existing programming concepts

7: The paradigm in which executable design is layered on top of the BPMN model, is **not** the way to go.

[Read the paper for more]

This slide show is based on two papers

Mapping BPMN to BPEL

http://eprints.qut.edu.au/5266/1/5266.pdf

Ouyiang, Dumas, van der Aalst and ter Hofstede

The Seven Fallacies of Business Process Execution

http://www.infoq.com/articles/seven-fallacies-of-bpm

Jean-Jacques Dubray

Avancier's conclusions from the experience recorded

- The BPM dream is reasonable in so far as:
  - Analysts use BPMN to visualize human activity system processes
  - Developers use BPEL to code computer activity system processes.

- However
  - A BPMN diagram is a cartoon for human actors
    - So don't try to transform BPMN diagrams into BPEL
  - BPEL is used by developers to code automated services
    - Which store, maintain and read data entities that represent the state data that must be remembered and tested to support and enable the business process

Architect Training and Methods from Avancier

Avancier

- Our training and methods are useful with all architecture frameworks that share similar domains and entities

- http://avancier.website