Business-Driven Software Engineering

Lecture 4 – Process Orchestration

Jochen Küster jku@zurich.ibm.com
Agenda

- Background on Workflow Patterns
- Control Flow Patterns
- Data Patterns
- Summary and References
Background on Workflow Patterns
Workflow Patterns

- control flow pattern represent reoccurring solutions how control flow in process models is specified
- data patterns for understanding how data is handled in process models
- resource patterns how resources are handled
- exception handling patterns

- independent of concrete process modeling languages
- for comparing expressiveness of process modeling languages
- represent common knowledge extracted from many process modeling languages
- www.workflowpatterns.com for more information
Control Flow Patterns
Overview of Patterns (adapted from W. van der Aalst)

Basic Control Flow Patterns
- Pattern 1 (Sequence)
- Pattern 2 (Parallel Split)
- Pattern 3 (Synchronization)
- Pattern 4 (Exclusive Choice)
- Pattern 5 (Simple Merge)

Advanced Branching/Synchronization Patterns
- Pattern 6 (Multi-Choice)
- Pattern 7 (Synchronizing Merge)
- Pattern 8 (Multi-Merge)
- Pattern 9 (Discriminator)

Structural Patterns
- Pattern 10 (Arbitrary Cycles)
- Pattern 11 (Implicit Termination)

Multiple Instances Patterns
- Pattern 12 – Pattern 15

State-based Patterns
- Pattern 16 (Deferred Choice)
- Pattern 17 (Interleaved Parallel Routing)
- Pattern 18 (Milestone)

Cancellation Patterns
- Pattern 19 (Cancel Activity)
- Pattern 20 (Cancel Case)
Control Flow Pattern - Sequence

- an activity is enabled after the termination of another activity
- used to model two consecutive steps in a process
- *Grant Claim* can only be enabled after *Register Claim* has terminated
- possible trace: <…Register Claim, Grant Claim,…>
Control Flow Pattern – And Split (Parallel Split)

- A single thread of control is split into multiple threads of control which are executed in parallel.
- After termination of Register Claim, both Grant Claim and Reject Claim are enabled.
- Activities Grant Claim and Reject Claim execute concurrently.
- Possible traces:
  - <Register Claim, Grant Claim, Reject Claim, …>
  - <Register Claim, Reject Claim, Grant Claim, …>
Control Flow Pattern – And Join (Synchronization)

- multiple threads of control are joined into one thread of control
- Close Claim is enabled only after both Grant Claim and Reject Claim have terminated
- used for joining concurrently executing branches
And Join – Traces and Activations

- possible traces:
  - ⟨..., Grant Claim, Reject Claim, Close Claim,…⟩
  - ⟨..., Reject Claim, Grant Claim, Close Claim,…⟩

- possible activation:
Exercise: Work out one trace and one activation

<RC,EC,EF,GC,DF,CC>
Control Flow Pattern – Xor Split (Exclusive Choice)

- based on a choice, one of several branches is chosen
- after termination of Register Claim, either Grant Claim or Reject Claim is enabled
- activities Grant Claim and Reject Claim cannot execute concurrently
- possible traces:
  - <..., Register Claim, Grant Claim, ...>
  - <..., Register Claim, Reject Claim, ...>
Control Flow Pattern – Xor Join (Simple Merge)

- two or more alternative branches are joined without synchronization
- only one of the alternative branches is executed (Assumption)
- possible trace:
  - <...Grant Claim, Close Claim,...>
  - <...Reject Claim, Close Claim,...>
Control Flow Pattern – Or Split (Multi Choice)

- one or more branches are enabled after the termination of the Register Claim activity
- can lead to concurrent execution, but need not to
- possible traces:
  - <…Register Claim, Grant Claim, Reject Claim,…>
  - <…Register Claim, Reject Claim,…>
  - and many more
Control Flow Pattern – Or Join (Synchronizing Merge)

- multiple concurrent threads are joined into one single thread
- already activated branch cannot be activated again while the Or Join is waiting for other branches (Assumption)
- problematic, because local decision is impossible: how to know how many branches have been activated and need to be waited for
- can lead to a deadlock situation if the Or Join is waiting forever
- solutions such as Dead-Path Elimination have been developed for implementing the Or-Join in Process Engines
Exercise: Work out traces and explain how Or Join works

<RC,GC,CC>  Or Join does not wait
<RC,RJ,CC>  Or Join does not wait
<RC,GC,RJ,CC>  Or Join waits
<RC,RJ,GC,CC>  Or Join waits
Control Flow Pattern – Multi Merge

- multiple branches reconverge without synchronization
- the activity following the merge is started for every activation of every incoming branch
- sometimes the Multi Merge is realized using the simple merge
- possible traces:
  - \(<RC,GC,CC,RJ,CC>, <RC,RJ,CC,GC>,...\)
Exercise: Work out one trace and one activation

<RC,EC,EF,GC1,DF1,GC2,DF2,CC1,CC2>
Control Flow Pattern - Discriminator

- discriminator waits for one of the incoming branches to complete before activating the following activity
- from that moment on it waits for remaining branches to complete and ignores them
- after all incoming branches have been triggered, it resets itself so that it can be triggered again (for loops)
arbitrary cycles allow the repetition of activities

be careful by jumping into parallel fragments

arbitrary cycles can lead to infinite processes

possible traces:
  – <Register Claim, Grant Claim, Close Claim, Close Claim, …>
Multiple Instances Without Synchronization

- a number of instances are created and started
- following activity can be enabled immediately after enabling the multiple instances
- activity instances are not synchronized
- example:
  - *Check Claim Detail* is performed for each detail in the claim
Multiple Instances With A Priori Design Time Knowledge

- number of activity instances is known at design time
- activity instances are synchronized
- example:
  - it is known that *Check Claim Detail* has to be performed two times
Multiple Instances With A Priori Run Time Knowledge

- number of activity instances is known at runtime, before the instances are created
- activity instances are synchronized
- example:
  - it is known that *Check Claim Detail* has to be performed n times
  - n depends on the number of registered Claim details
- Multiple instances without a priori run time knowledge (not discussed further)
Control Flow Pattern – Deferred Choice

- deferred choice represents a point in a process where a decision is made based on the environment
- the process offers several alternatives to the environment and then, depending on the state of the environment, a choice is made
- the moment of time for the decision is “deferred” to the latest point possible
- in the example, we assume that either branch is activated by receiving a message from the environment
Control Flow Pattern – Interleaved Parallel Routing

- better name: sequential execution without a priori design time knowledge
- a set of activity instances is executed sequentially
- order of execution is decided at runtime
- possible trace:
  - <Register Claim, Grant Claim, Reject Claim, Close Claim,...>
Exercise: Work traces and activations

<RC,EC,EF,GC,CC>, <RC,EF,EC,RC,CC>, …
Exercise: Work out traces and activations

<RC, EC, GC, RC, CC>, <RC, EF, EC, RC, GC, CC>, …
Implicit Termination:

- a given process instance should be terminated if nothing else is to be done
- represents a termination condition for process instances
- in contrast to explicit termination by reaching a final node
- no explicit representation required (because it is implicit)

Milestone:

- Used for expressing that an activity is only enabled if a certain milestone has been reached and is not expired
- Not supported in most process modeling languages
Control Flow Patterns – Not discussed in detail

Cancel activity:

- An enabled activity is disabled
- The activity enters the *cancelled* state

Cancel case:

- All activity instances of a process instance are cancelled
## Comparison of Workflow Languages (Example)

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Staffware</th>
<th>WebSphere MQ</th>
<th>FLOWer</th>
<th>COSA</th>
<th>iPlanet</th>
<th>SAP Workflow</th>
<th>FileNet</th>
<th>BPEL</th>
<th>WebSphere BPEL</th>
<th>Oracle BPEL</th>
<th>BPMN</th>
<th>XPDL</th>
<th>UML ADL</th>
<th>EPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (seq)</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>2 (par-epl)</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>3 (synch)</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>4 (ex-cl)</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>5 (simple-m)</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>6 (m-choice)</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>7 (s-synt-m)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>8 (multi-m)</td>
<td>-</td>
<td>-</td>
<td>+/-</td>
<td>+/-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>9 (s-disc)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+/-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+/-</td>
<td>+/-</td>
<td>+/-</td>
<td>+/-</td>
<td>-</td>
</tr>
<tr>
<td>10 (act-c)</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>11 (impl-c)</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>12 (mi-no-s)</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+/-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>13 (mi-dt)</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>14 (mi-rt)</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>15 (mi-no)</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>16 (def-c)</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+/-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>17 (mi-pac)</td>
<td>-</td>
<td>-</td>
<td>+/-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+/-</td>
<td>+/-</td>
<td>+/-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>18 (milest)</td>
<td>-</td>
<td>-</td>
<td>+/-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>19 (can-a)</td>
<td>+</td>
<td>-</td>
<td>+/-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>20 (can-c)</td>
<td>-</td>
<td>-</td>
<td>+/-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

Data Patterns
Data Patterns

- patterns that capture in which data is represented and used in workflows

- four groups of data patterns:
  - data visibility: how can data elements be viewed in the process
  - data interaction: how is data communicated in the process
  - data transfer: how is data transferred within workflow components
  - data-based routing: how can data elements influence e.g. the control flow view

- in the following:
  - selected data patterns
  - no data transfer patterns
Data Visibility – Activity/Block Data

Pattern 1 (Activity Data)
- data elements can be defined by activities. Data elements are accessible only within the context of the execution of activity instances.

Pattern 2 (Block Data)
- data elements are defined in a block (e.g. composite activity). Data elements are accessible by each of the activities in the block.
Pattern 3 (Case Data)
- data elements are supported that are specific to a process instance. All components of the process instance can access them.

Pattern 4 (Process Data)
- data elements are supported that are specific to a process. All components in all process instances can access them.
Data Interaction – Activity to Activity

Pattern 5 (Activity to Activity)

- a data element can be communicated from one activity to another in a process instance.
Data Interaction – to multiple instance task

Pattern 6 (to multiple instance task)

- data elements can be passed from a preceding activity instance to a subsequent activity that supports multiple execution instances.
- involves either passing all data elements to each of the following instances or distributing them
Pattern 7 (from multiple instance task)

- data elements can be passed from an activity that supports multiple instances to a subsequent activity.
- data elements can be aggregated.
Pattern 8 (Case to Case)

- data elements can be passed from one process instance to another process instance while they are concurrently executing
- allows one process instance to access results of another process instance
Pattern 9 (Activity Precondition – Data Existence)

- data-based preconditions are specified for activity based on the presence of data elements at the time of execution
- only execute Grant Claim if Claim is available
- defer activity execution or skip activity
- request values from the user or use default values
- kill the process instance
Pattern 10 (Activity Precondition – Data Value)

- data-based preconditions can be specified for activities based on the value of specific data elements at the time of execution
- only execute Grant Claim if Claim is in “registered” state
- skip activity or delay activity execution
Pattern 11 (Activity Postcondition – Data Existence)

- data-based postconditions can be specified for activities based on the existence of specific data elements at the time of execution.
- do not complete Settle Claim before a Settlement has been created
- usually implies an implicit loop of the activity, another execution of the activity
Data-based Routing

Pattern 12 (Activity Postcondition – Data Value)

- data-based postconditions can be specified for activities based on the value of specific data elements at the time of execution.

- similar to previous pattern
Data-based Routing

Pattern 13 (Event-based ActivityTrigger)

- the ability for an external event to initiate an activity.
- activity is waiting for an event to occur for resuming execution
- resume Settle Claim once Customer is Ready
Data-based Routing

Pattern 14 (Data-based Task Trigger)
- the ability to trigger a specific task when an expression based on data elements evaluates to true.
- Settle Claim is started once the Customer state is in “Ready”
Pattern 15 (Data-based Routing)

- the ability to alter the control flow as a consequence of the value of data-based expressions
- examples: exclusive choice and multi-choice
- perform the decision based on the state of the Claim
Summary of Lecture and References

- discussion of control flow pattern
- discussion of data patterns
- valuable for comparing expressiveness of workflow systems and semantics of process modeling languages

Further Reading:

- M. Weske: Chapter 4
- original publications: