Implementing Workflow Patterns with Business Process Choreography
Workflow Patterns – Objectives

- Describe workflow patterns implementation
  - The patterns themselves (conceptual)
  - Implementation of these patterns in BPEL4WS & BPC v5.1
  - Equivalent implementations/options in BPEL4WS & BPC v5.1
  - Differences between Long-running & Non-interruptible processes

- After this presentation you should...
  - Be able to recognize simple patterns in business processes
  - Be able to use this presentation and its samples as a jump-start to implement specific business processes
Workflow Patterns – Agenda

- Terms
- Workflow Patterns
  - Basic Control Flow Patterns (5)
  - Structural Patterns (2)
  - Cancellation Patterns (2)
  - Communication Patterns (2)
  - State based Patterns (3)
  - Advanced Branching and Synchronization Patterns (5)
  - Patterns with multiple instances (4)
- Summary
Workflow Patterns – Terms

- Synonyms, Abbreviations & Similar meanings
  - Branch
    - Path or Execution Path
    - Thread of Control
  - Long-Running Process (LR)
  - Non-Interruptible (NI)
  - Synchronous interface
    - Request - Response
    - Input - Output
  - Asynchronous interface
    - Call – Callback
    - Fire and Forget
Workflow Patterns

- Basic Control Flow Patterns (5)
  - Sequence
  - Parallel Split
  - Synchronization
  - Exclusive Choice
  - Simple Merge

- Structural Patterns (2)
- Cancellation Patterns (2)
- Communication Patterns (2)
- State based Patterns (3)
Sequence

- **Context**
  - Activities may depend on other activities

- **Problem**
  - Some activities may have to wait until other activities complete

- **Solution**
  - Use sequence
  - Or flow
    - plus links or joinConditions that evaluate to true
Sequence Example
Parallel Split

- **Context**
  - Activities have no direct dependencies and parallel execution is required (LR)

- **Problem**
  - A single thread of control has to split into 2 or more different execution paths (branches)

- **Solution**
  - Use *links*
  - Or *flow*
Parallel Split

see next slide...
Synchronization

- **Context**
  - Multiple branches need to converge into a single execution path \((LR)\)

- **Problem**
  - Synchronize parallel execution paths

- **Solution**
  - Use *links* in *flows*
  - Or continue after *flow*
Exclusive Choice

- **Context**
  - Different paths need to be chosen based on a condition

- **Problem**
  - Decide which path to go and trigger one path only

- **Solution**
  - Use `switch`
Simple Merge (1)

- **Context**
  - There are multiple execution paths, only one is executed

- **Problem**
  - We need a merge point

- **Solution**
  - Use `switch` within a `sequence`
  - Or, use `switch` within a `flow` and continue with `link`
Simple Merge (2)

- If modeled with exclusive conditions this flow results in the same behavior as the previous one.

- Note: there are several options to model a specific pattern:
  - choose the simplest one
    - to design
    - to understand
Workflow Patterns

- Basic Control Flow Patterns (5)
- Structural Patterns (2)
  - Arbitrary Cycles
  - Implicit Termination
- Cancellation Patterns (2)
- Communication Patterns (2)
- State based Patterns (3)
Arbitrary Cycles

- **Context**
  - A set of *activities* needs to be executed multiple times

- **Problem**
  - Specific entry points are necessary as starting points for the execution of a path
  - Exiting the path requires additional activities or information

- **Solution**
  - You cannot realize this pattern with BPEL4WS; however, structured cycles are possible with *while activities*
Implicit Termination

- **Context**
  - The workflow engine (BP Engine) must determine when a process has ended

- **Problem**
  - There may be processes that have no *reply* or *terminate activity*

- **Solution**
  - Implicit termination is provided with BPEL4WS, *terminate* is not required to end a process
Workflow Patterns

- Basic Control Flow Patterns (5)
- Structural Patterns (2)
- Cancellation Patterns (2)
  - Cancel Activity
  - Cancel Case
- Communication Patterns (2)
- State based Patterns (3)
Cancel Activity

- **Context**
  - Some scenarios require that *activities* are disabled for some reason

- **Problem**
  - Skip *activities* based on conditions

- **Solution**
  - Use *flow* and
    - Decide with *links* & its conditions
    - Let the *join condition* decide
  - Use *switch*
Cancel Case

- **Context**
  - The execution of a process must be stopped. Running activities will be stopped immediately as well.

- **Problem**
  - There may be sub-processes running that may need to be stopped as well
  - Only child sub-processes will be stopped

- **Solution**
  - Use `terminate`
Workflow Patterns – Agenda

- Basic Control Flow Patterns (5)
- Structural Patterns (2)
- Cancellation Patterns (2)
- Communication Patterns (2)
  - Request/Reply
  - One-Way
- State based Patterns (3)
Request / Response & One-Way

- **Context**
  - Communication between processes, talking to web services

- **Problem**
  - Use of input/output and input-only operations with BPEL4WS

- **Solution**
  - See picture
Workflow Patterns

- Basic Control Flow Patterns (5)
- Structural Patterns (2)
- Cancellation Patterns (2)
- Communication Patterns (2)
- State based Patterns (3)
  - Deferred Choice
  - Interleaved Parallel Routing
  - Milestone
Deferred Choice \((LR)\)

- **Context**
  - The modeler cannot model the process such that a particular branch can be chosen based on the data available to the process.

- **Problem**
  - A particular branch must be chosen (compare *choice pattern*) based on a decision not influenced by the process model.

- **Solution**
  - Use *pick*
Interleaved Parallel Routing \(^{(NI)}\)

- **Context**
  - A set of activities needs to be executed – no concurrent active activities are allowed

- **Problem**
  - The activities should be executed in an arbitrary order that will be determined at runtime

- **Solution**
  - Use *non-interruptible* processes and the *flow*
  - Or *non-interruptible* processes and *links*
Milestone

- **Context**
  - An Activity is enabled after a FirstActivity completed as long as it completes is a certain amount of time

- **Problem**
  - Disable Activity after time has expired with Invoke

- **Solution**
  - Use pick (LR) and send end event in Invoke after the timer.
Workflow Patterns

- Advanced Branching and Synchronization (5)
  - Multi Choice
  - Synchronizing Merge
  - Multi-Merge
  - Discriminator
  - N-out-of-M Join
- Patterns with multiple instances (4)
Multi Choice

- **Context**
  
  - There are a number of branches that may need to be executed based upon a condition that isn’t determined until run time

- **Problem**
  
  - Decide which branches to execute, possibly fork (LR)

- **Solution**
  
  - Use *links* and apply conditions
Synchronizing Merge

- **Context**
  - Parallel path synchronization, not all paths are necessarily executed

- **Problem**
  - Synchronize parallel execution (LR) paths and ignore skipped execution paths

- **Solution**
  - Use *links* in *flow*
  - Or continue after *flow*
Multi Merge (1)

- **Context**
  - For a set of different branches the same service must be called

- **Problem**
  - Multiple alternatives must be executed based upon a condition. After executing the alternatives a special service must be called

- **Solution**
  - There is no special activity for that; call the specific service after each path

**Diagram**

- Sequence
- Order
- Add CD
- Possibly another process
Multi Merge (2)

asynchronous activation of service
Discriminator without Sync (LR)

- **Context**
  - A set of concurrent branches is executed – continue as soon as the fastest branch is complete

- **Problem**
  - Guarantee that the succeeding activity is executed only once; continue without waiting for other branches

- **Solution**
  - Use *receive* or *pick*
Discriminator with Sync (LR)

- **Context**
  - A set of concurrent branches is executed – next activity must be started as soon as possible and only once (after first branch is completed)

- **Problem**
  - In addition to “Discriminator without Sync”, synchronization needs to be done after all branches have been executed

- **Solution**
  - Use `receive` in a loop
N-out-of-M Join (LR)

- **Context**
  - Multiple alternatives must be executed based upon a condition and must synchronize again

- **Problem**
  - Execution of an activity in the workflow must happen as soon as N out of M paths are completed

- **Solution**
  - Use `receive` in a loop
Workflow Patterns

- Advanced Branching and Synchronization Patterns (5)
- Patterns with multiple instances (4)
  - without synchronization
  - with a priori design time knowledge
  - with a priori runtime knowledge
  - without a priori runtime knowledge
Multiple Instances without Synchronization

**Context**
- An activity must be started multiple times; however, control over the execution of the activity is not necessary.

**Problem**
- For each instance of an *activity* an additional thread of control must be started.\(^{(LR)}\)

**Solution**
- Use *while activity* and call another process asynchronously.
Multiple Instances with design time knowledge

- **Context**
  - The number of instances of an *activity* are known at design time; synchronization after execution is needed

- **Problem**
  - Multiple threads of control need to be started (LR) and synchronized after execution

- **Solution**
  - use *flow* and model the number of *activities*
Multiple Instances with runtime knowledge

**Context**
- The number of instances of an activity is known at runtime

**Problem**
- Multiple instances must be started in parallel (LR) – asynchronously, then synchronized again

**Solution**
- See picture
Multiple Instances without runtime knowledge

- **Context**
  - See pattern on previous page; the number of instances is not known at runtime

- **Problem**
  - Concurrently start an unknown number of activities and gain control again (LR) for synchronization

- **Solution**
  - See picture
Summary

- Reviewed simple workflow patterns
  - The patterns themselves (conceptual)
  - Implementation of these patterns in BPEL4WS & WPC v5.1

- Discussed several equivalent implementations of certain patterns in BPEL4WS & WPC v5.1

- Pointed out differences between Long-running & Non-interruptible processes and how they effect the patterns

- Built a foundation to discuss more advanced workflow patterns