Обзор – Перспективы WorkFlow-систем
Применение WF- паттернов для сравнения WF-языков.

Приложение к статье "Сравнение workflow-языков".

Андрей Михеев, Михаил Орлов

В настоящем приложении к статье "Сравнение workflow-языков" на примерах реализаций некоторых характерных участков бизнес-процессов (WF-паттернов) показаны основные конструкции распространенных языков управления бизнес-процессами (WF-языков), относящиеся к перспективе потока управления.

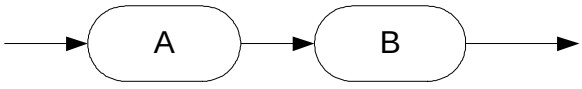
- BPEL (Коалиция IBM, Microsoft, BEA, SAP и Siebel)
- BPML (Коалиция BPMI)
- XPDL (Коалиция WfMC)
- JPDL (проект JBOSS JBPM)

Первые три языка соответствуют различным стандартам международных коалиций, четвертый язык приведен как язык, ориентирующийся на поддержку WF-паттернов.

Приведенные в приложении примеры показывают, что при реализации WF-паттернов на различных языках конкретный синтаксис выражений отличается, но основные конструкции каждого языка можно отнести к одному из двух классов (либо к граф-ориентированному классу, либо к структурно-ориентированному).

Паттерн "последовательность".

Описание. Два узла ("A" и "B") соединены переходом. После выполнения Действия узла "A" управление переходит узлу "B".



Реализация паттерна "последовательность" на языке XPDL

```
<WorkflowProcess Id=”1” Name=”PatternSequence”>
     …
     <Activities>
          <Activity Id=”1” Name=”A”>
               …
          </Activity>
          <Activity Id=”2” Name=”B”>
               …
          </Activity>
     </Activities>
     <Transitions>
          <Transition Id=”1” From=”1” To=”2” Name=”FromAtoB“/>
     </Transitions>
</WorkflowProcess>
```

Реализация паттерна “последовательность” на языке BPML

```
<process name=”PatternSequence”>
     <sequence>
          <action name=”A” …>
               …
          </action>
          <action name=”B” …>
               …
          </action>
     </sequence>
</process>
```

Реализация паттерна “последовательность” на языке BPEL4WS

Без использования конструкции “Link”

```
<process name=”PatternSequence”…>
     …
     <sequence>
          <receive … operation=”operation A” …>
               …
          </receive >
          < receive … operation=”operation B” …>
               …
          </receive >
     </sequence>
</process>
```

С использованием конструкции “Link”

```
<process name=”PatternSequence”…>
     …
     <flow>
          <links>
               <link name=”FromAtoB”>
          </links>
          <receive … operation=”operation A” …>
               …
               <source linkName=”FromAtoB/>
          </receive >
          < receive … operation=”operation B” …>
               …
               <target linkName=”FromAtoB/>
          </receive >
     </flow>
</process>
```

Реализация паттерна "последовательность" на языке jPdl

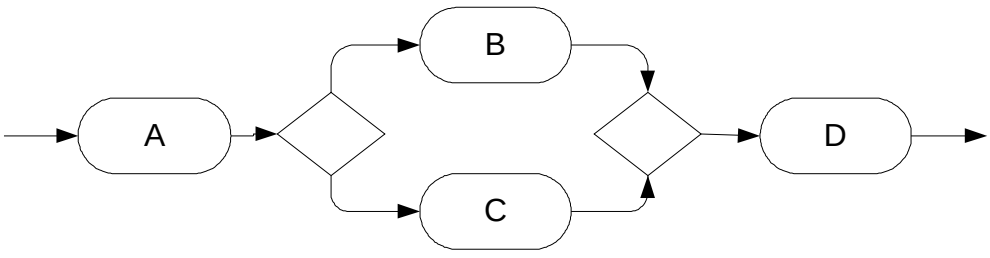
```
<process-definition name=" PatternSequence ">
     …
     <state name="A">
          …
          <transition to="B">
          </transition>
     </state>

     <state name="B">
          …
          <transition to="…">
          </transition>
     </state>
     …
</process-definition>
```

Паттерны "исключающий выбор" и "простое соединение".

Описание паттерна "исключающий выбор". Точка в графе workflow-процесса (узел), в которую приходит только один переход и из которой исходит два или более переходов. Причем, при выполнении действия узла существует только один поток управления бизнес - процессом, который соответствует одному из исходящих переходов.

Описание паттерна "простое соединение". Точка в графе workflow-процесса (узел), в которой соединяются два или более перехода, а выходит только один переход. После выполнения действия узла существует только один поток управления, соответствующий выходящему переходу. Предполагается, что только по одному из входящих переходов в узел может придти управление.

Описание реализуемой конструкции. Связка паттернов "исключающий выбор" и "простое соединение". Управление находится в узле "A", далее управление переходит либо в узел "B", либо в узел "C", в зависимости от выполнения некоторого условия, далее из любого из этих узлов управление переходит в узел "D":



Реализация паттернов “исключающий выбор” и “простое соединение” на языке XPDL

```
<WorkflowProcess Id=”1” Name=”PatternsExclusiveChoiceSimpleMerge”>
     …
     <Activities>
          <Activity Id=”1” Name=”A”>
               …
               <Transition Restrictions>
                    <Transition Restriction>
                         <Split Type=”XOR”>
                              <TransitionRefs>
                                   <TransitionRef Id=”1”/>
                                   <TransitionRef Id=”2”/>
                              </TransitionRefs>
                         </Split>
                    </Transition Restriction>
               </Transition Restrictions>
          </Activity>
          <Activity Id=”2” Name=”B”>
               …
          </Activity>
          <Activity Id=”3” Name=”C”>
               …
          </Activity>
          <Activity Id=”4” Name=”D”>
               …
               <Transition Restrictions>
                    <Transition Restriction>
                         <Join Type=”XOR”/>
                    </Transition Restriction>
               </Transition Restrictions>
          </Activity>
     </Activities>
     <Transitions>
          <Transition Id=”1” From=”1” To=”2” Name=”FromAtoB“>
               <Condition> … </Condition>
          </Transition>
          <Transition Id=”2” From=”1” To=”3” Name=”FromAtoC“>
               <Condition> … </Condition>
          </Transition>
          <Transition Id=”3” From=”2” To=”4” Name=”FromBtoD“/>
          <Transition Id=”4” From=”3” To=”4” Name=”FromCtoD“/>
     </Transitions>
</WorkflowProcess >
```

В данном примере мы видим, что в первой части описания процесса последовательно перечисляются узлы, а во второй части – переходы между узлами. Каждый переход соответствует упорядоченной паре: (ссылка на исходящий узел, ссылка на входящий узел). Понятно, что таким образом можно описать математический граф любой сложности.

С другой стороны, все описания узлов в XPDL "равноправны", описания переходов определяются независимо друг от друга, в языке можно задать переход между любыми двумя узлами – в этом языке очень сложно будет определить системы вложенных областей, в которые можно входить только через определенные узлы (нельзя входить через "боковые стороны"). То есть, такие конструкции, как бизнес-исключения (exceptions) и компенсации (compensations), очень удобные для многих реальных бизнес-процессов, в этом языке ввести сложно.

В данном примере мы также видим, что разветвления (Split) и слияния (Join) определяются внутри описаний узлов, как ограничения на входящие и исходящие переходы, а условия выбора конкретного перехода определяются в самом переходе.

Реализация паттернов "исключающий выбор" и "простое соединение" на языке BPML

```
<process name="PatternsExclusiveChoiceSimpleMerge">
     <sequence>
          <action name="A" …>
               …
          </action>
          <switch>
               <case>
                    <condition> … </condition>
                    <action name="B" …>
                         …
                    </action>
               </case>
               <case>
                    <condition> … </condition>
                    <action name="C" …>
                         …
                    </action>
               </case>
               …
          </switch>
          <action name="D" …>
               …
          </action>
     </sequence>
</process>
```

Данный вариант записи бизнес-процесса существенно короче и легче для понимания, чем вариант на языке XPDL. В значительной степени, это следствие того, что в BPML вообще не употребляется конструкция "переход между узлами". Если нет специальных управляющих тегов, то узлы выполняются последовательно в порядке их расположения в описании процесса.

Тег switch определяет выбор для выполнения единственного из перечисленных в нем узлов (по аналогии с языками программирования, в зависимости от выполнения условия тега condition). В языке BPML также существует тег, определяющий одновременный порядок выполнения всех находящихся в нем узлов (тег All). Комбинирование этих тегов

позволяет создавать достаточно мощные конструкции из вложенных параллельных и последовательных элементов разных типов.

Однако, как известно из математики, описать таким образом граф произвольной структуры нельзя. В частности, нельзя таким образом задать нерегулярные циклы – такие циклы, входить и выходить в которые можно через "боковые стороны". Это достаточно серьезное ограничение всех структурно-ориентированных WF-языков.

С другой стороны, конструкция из вложенных и последовательных тегов хорошо подходит для определения в них таких элементов, как exceptions (исключения) и compensations (компенсации).

Реализация паттернов "исключающий выбор" и "простое соединение" на языке BPEL4WS

```
<process name="PatternsExclusiveChoiceSimpleMerge"…>
     …
     <sequence>
          <receive … operation="operation A" …>
               …
          </receive >
          <switch…>
               <case condition=…>
                    <receive … operation="operation B" …>
                         …
                    </receive >
               </case>
               <case condition= …>
                    <receive … operation="operation C" …>
                         …
                    </receive >
               </case>
          </switch…>
          < receive … operation="operation D" …>
               …
          </receive >
     </sequence>
</process>
```

Мы видим, что данная реализация примера паттернов "исключающий выбор" – "простое соединение" практически повторяет реализацию на языке BPML (с точностью до названия тегов). BPEL, однако, допускает второй вариант реализации примера – с использованием конструкции link:

```
<process name=" PatternsExclusiveChoiceSimpleMergeWithLink"…>
     …
     <flow>
          <links>
               <link name="FromAtoB">
               <link name="FromAtoC">
               <link name="FromBtoD">
               <link name="FromCtoD">
          </links>
```

```
          <receive … operation=”operation A” …>
               …
               <source linkName=”FromAtoB transitionCondition=”…”/>
               <source linkName=”FromAtoC transitionCondition=”…”/>
          </receive >

          <receive … operation=”operation B” …>
                    …
               <target linkName=”FromAtoB/>
               <source linkName=”FromBtoD/>
          </receive >

          <receive … operation=”operation C” …>
                    …
               <target linkName=”FromAtoC/>
               <source linkName=”FromCtoD/>
          </receive >

          < receive … operation=”operation D” …>
               …
               <target linkName=”FromBtoD/>
               <target linkName=”FromCtoD/>
          </receive >
     </flow>
</process>
```

В этом примере все узлы бизнес-процесса помещены внутрь тега параллельного выполнения “flow”, однако часть узлов связана отношениями:

- Выполнить перед – соответствует тегу “source”
- Выполнить после – соответствует тегу “target”

Видно, что второй вариант реализации длиннее и менее понятен, но иногда использование конструкции link имеет преимущества.

Замечание. В примере реализации на языке BPML использован тег action, который подразумевает “активность” самого бизнес-процесса, т.е. бизнес-процесс сам “дает задания” web-сервисам – исполнителям и ждет результатов их работы. В примере реализации на языке BPEL4WS использован “пассивный” тег receive: после того, как управление попадает в этот тег, бизнес-процесс останавливается и, не выполняя никаких активных действий, ждет, пока определенный web-сервис пришлет ему сообщение. Данный тег был использован в примере для того, чтобы еще раз подчеркнуть, что языки BPML и BPEL4WS допускают как “активное”, так и “пассивное” поведение узлов, в отличие от XPDL, в котором все узлы только “пассивные”. Для “активного” поведения узла в BPEL4WS надо использовать тег Invoke.

Реализация паттернов “исключающий выбор” и “простое соединение” на языке jPDL

```
<process-definition name="PatternsExclusiveChoiceSimpleMerge">
     …
     <state name="A">
          …
          <transition to="Decision”/>
     </state>
     <decision name=" Decision">
          …
          <delegation …>
               …
               if(…) return "approve”;
               else return " disapprove”;
               …
          </delegation>
          <transition name="approve"     to="B"/>
          <transition name="disapprove"  to="C"/>
     </decision>
     <state name="B">
          …
          <transition to="D">
          </transition>
     </state>
     <state name="C">
          …
          <transition to="D">
          </transition>
     </state>
     <state name="D">
          …
          <transition to="…">
          </transition>
     </state>
     …
</process-definition>
```

Конструкции языка jPDL, связанные с переходами, в отличие от XPDL, описываются не как отдельные сущности, “равноправные” с узлами графа бизнес-процесса, а только как вложенные теги относительно узлов графа. Вследствие этого, эти конструкции существенно “легче” аналогичных конструкций языка XPDL. В целом, язык jPDL значительно проще XPDL.

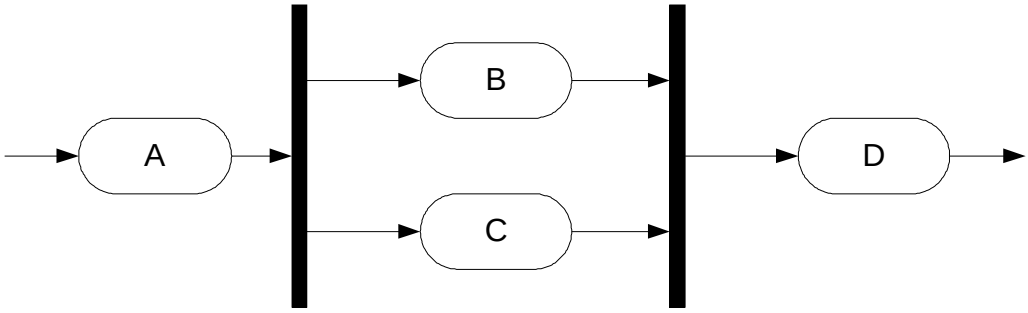
Паттерны “параллельное расщепление” и “синхронизация”.

Описание паттерна “параллельное расщепление”. Точка в графе workflow-процесса (узел), в которую приходит только один переход и из которой исходит два или более переходов. Причем, после выполнения действия узла поток управления бизнес-процессом распадается на количество потоков, равное количеству исходящих переходов. Далее потоки, соответствующие исходящим переходам, выполняются параллельно.

Описание паттерна “синхронизация”. Точка в графе workflow-процесса (узел), в которой соединяются два или более перехода, а выходит только один переход. Действие узла недоступно для выполнения, пока управление всех потоков управления бизнес-процессом (количество потоков равно количеству входящих переходов) не перейдет в данный узел. После выполнения действия узла существует только один поток управления, соответствующий выходящему переходу.

Предполагается, что по всем входящим переходам в узел придет управление, причем не может быть, чтобы по какому-либо направлению управление пришло более одного раза до выполнения действия узла.

Описание реализуемой конструкции: Связка паттернов “параллельное расщепление” и “синхронизация”. Управление находится в узле “A”, далее управление переходит одновременно в узел “B” и в узел “C”, после выполнения действий обоих узлов управление переходит в узел “D”:



Реализация паттернов “параллельное расщепление” и “синхронизация” на языке XPDL

Рассматривается следующая конструкция: управление находится в узле “A”, после того, как выполнено Действие узла A, поток управления распадается на два параллельных потока управления. Первый поток управления переходит в узел “B”, одновременно второй поток управления переходит в узел “C”. После выполнения Действий узлов “B” и “C” потоки управления сливаются в один, и этот поток управления переходит в узел “D”:

```
<WorkflowProcess Id=”1” Name=”PatternsParallelSplitSynchronization”>
     …
     <Activities>
          <Activity Id=”1” Name=”A”>
          …
               <Transition Restrictions>
                    <Transition Restriction>
                         <Split Type=”AND”>
```

```
                              <TransitionRefs>
                                   <TransitionRef Id=”1”/>
                                   <TransitionRef Id=”2”/>
                              </TransitionRefs>
                         </Split>
                    </Transition Restriction>
               </Transition Restrictions>
          </Activity>
               <Activity Id=”2” Name=”B”>
               …
          </Activity>
               <Activity Id=”3” Name=”C”>
               …
          </Activity>
          <Activity Id=”4” Name=”D”>
          …
               <Transition Restrictions>
               <Transition Restriction>
               <Join Type=”AND”/>
                    </Transition Restriction>
               </Transition Restrictions>
          </Activity>
     </Activities>
     <Transitions>
          <Transition Id=”1” From=”1” To=”2” Name=”FromAtoB“>
               <Condition> … </Condition>
          </Transition>
          <Transition Id=”2” From=”1” To=”3” Name=”FromAtoC“>
               <Condition> … </Condition>
          </Transition>
          <Transition Id=”3” From=”2” To=”4” Name=”FromBtoD“/>
          <Transition Id=”4” From=”3” To=”4” Name=”FromCtoB“/>
     </Transitions>
</WorkflowProcess >
```

Реализация паттернов “параллельное расщепление” и “синхронизация” на языке BPML

```
<process name=”PatternsParallelSplitSynchronization”>
     <sequence>
          <action name=”A” …>
          …
          </action>
          <all>
               <action name=”B” …>
               …
               </action>
               <action name=”C” …>
               …
               </action>
          </all>
          <action name=”D” …>
          …
          </action>
     </sequence>
</process>
```

Реализация паттернов “параллельное расщепление” и “синхронизация” на языке BPEL

```
<process name=”PatternsParallelSplitSynchronization”…>
     …
     <sequence>
          <receive … operation=”operation A” …>
               …
          </receive >
          <flow>
               <receive … operation=”operation B” …>
                    …
               </receive >
               <receive … operation=”operation C” …>
                    …
               </receive >
          </flow>
          < receive … operation=”operation D” …>
               …
          </receive >
     </sequence>
</process>
```

Реализация паттернов “параллельное расщепление” и “синхронизация” на языке jPdl:

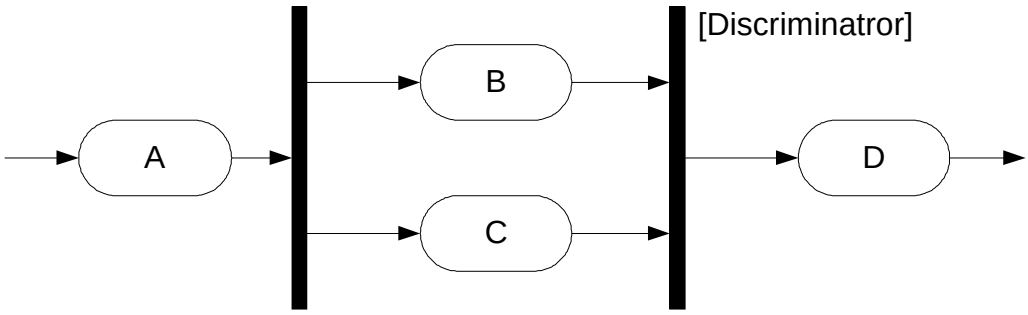
```
<process-definition name="PatternsParallelSplitSynchronization">
     …
     <state name="A">
          …
          <transition to="Fork”/>
     </state>
     <fork name="Fork" corresponding-join="Join">
          <transition to="B" />
          <transition to="C" />
     </fork>
     <state name="B">
          …
          <transition to="Join">
          </transition>
     </state>
     <state name="C">
          …
          <transition to="Join">
          </transition>
     </state>
     <join name="Join">
          <transition to="D" />
     </join>
     <state name="D">
          …
          <transition to="…">
          </transition>
     </state>
     …
</process-definition>
```

Паттерн “дискриминатор”.

Описание паттерна “дискриминатор”. Точка в графе workflow-процесса (узел), в которой соединяются два или более перехода, а выходит только один переход. Для первого пришедшего в узел управления (по “своему” переходу) выполняется действие узла. Выполнение других активных потоков не прерывается, однако приход управления каждого из этих потоков в узел дискриминатора игнорируется, а поток в этот момент заканчивает свое существование.

Предполагается, что по всем активным входящим переходам в узел дискриминатора когда-нибудь придет управление, причем не может быть, чтобы по какому-либо направлению управление пришло более одного раза. Пока все активные потоки не достигнут узла дискриминатора, он не может быть еще раз активизирован (то есть вновь пришедшее управление (например, в цикле) в соответствующий дискриминатору узел, реализующий паттерн “параллельное расщепление”, должно ждать окончания всех потоков дискриминатора).

Описание реализуемой конструкции. Связка паттернов “параллельное расщепление” и “дискриминатор”. Управление находится в узле “A”, далее управление переходит одновременно в узел “B” и в узел “C”, после выполнения действия любого узла управление переходит в узел “D”, после выполнения действия оставшегося узла соответствующий поток управления завершается.



Исследования авторов workflow паттернов показали, что на языках XPDL и BPEL4WS нельзя реализовать паттерн “дискриминатор”, однако при помощи языка BPML паттерн “дискриминатор” реализовать можно. На языке jPdl паттерн “дискриминатор” также реализовать нельзя.

Реализация паттернов “параллельное расщепление” и “дискриминатор” на языке BPML

Рассматривается следующая конструкция: управление находится в узле “A”. После того, как выполнено Действие узла A, поток управления распадается на два параллельных потока управления. Первый поток управления переходит в узел “B”, одновременно второй поток управления переходит в узел “C”. После выполнения Действия узла “B” или “C” (того действия, которое выполнится раньше), поток управления переходит в узел “D”. В “оставшемся” узле продолжает находиться управление. После выполнения Действия этого узла эта точка управления перестает существовать.

```
<process name=”PatternDiscriminator”>
     <sequence>
          <context>
               <signal name=”completedBorC”/>
               <process name=”B”>
                    …
                    <raise signal=” completedBorC”/>
               </process>
               <process name=”C”>
                    …
                    <raise signal=” completedBorC”/>
               </process>
          </context>
          <action name=”A” …>
          …
          </action>
          <all>
               <spawn process=”B”/>
               <spawn process=”C”/>
          </all>
          <synch signal=” completedBorC”/>
          <action name=”D” …>
          …
          </action>
     </sequence>
</process>
```

Регулярный цикл.

Описание паттерна “произвольный цикл”. Участок в графе workflow-процесса (набор узлов), в котором один узел (или несколько узлов) может многократно выполняться, в общем случае в этот узел (набор узлов) управление может приходить из разных узлов, уходить управление тоже может в различные узлы.

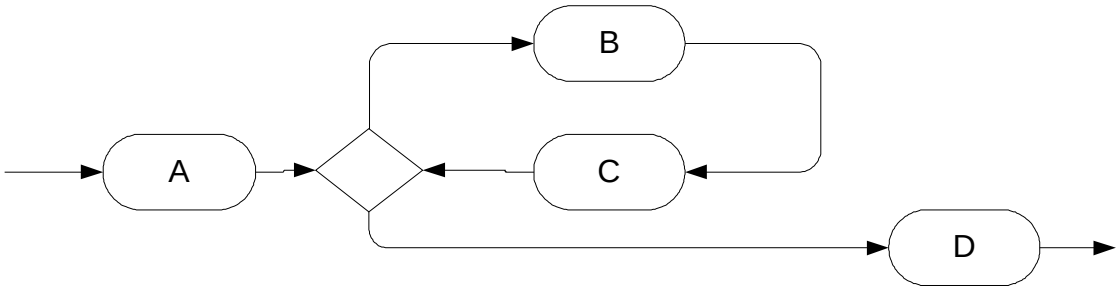
Возможность реализации паттерна “произвольный цикл” на языках XPDL и BPML

Так как XPDL – граф-ориентированный язык, реализация на нем паттерна “произвольный цикл” не вызывает никаких проблем. В случае BPML ситуация иная. В силу структурной ориентированности BPML, в нем невозможно описать переходы, нарушающие границы текущего блока тегов, то есть BPML не поддерживает паттерна “произвольный цикл”. Однако язык BPML допускает несколько видов регулярных циклов. В некотором смысле, подход языка BPML к циклам напоминает парадигму программирования “без использования оператора GOTO”. При помощи языка BPEL4WS также нельзя реализовать паттерн “произвольный цикл”, однако язык поддерживает регулярные циклы. На языке jPdl, в силу его граф-ориентированности, произвольный цикл реализовать можно.

Замечание. В данном разделе для простоты конструкций для всех WF-языков будет рассмотрен не произвольный, а только регулярный цикл.

Описание рассматриваемой конструкции – регулярного цикла: Управление находится в узле “A”. После того, как выполнено Действие узла “A”, происходит

следующий итеративный процесс: проверяется условие. Если оно истинно, то управление переходит в узел “B”, а после выполнения Действия узла “B” - в узел “C”. Далее итерации повторяются до тех пор, пока условие не станет ложным. После этого поток управления переходит в узел “D”.



Реализация регулярного цикла на языке XPDL

```
<WorkflowProcess Id=”1” Name=”PatternsExclusiveChoiceSimpleMerge”>
     …
     <Activities>
          <Activity Id=”1” Name=”A”>
               …
          </Activity>
          <Activity Id=”2” Name=”B”>
               …
               <Transition Restriction>
                    <Join Type=”XOR”/>
               </Transition Restriction>
          </Activity>
          <Activity Id=”3” Name=”C”>
               …
               <Transition Restrictions>
                    <Transition Restriction>
                         <Split Type=”XOR”>
                              <TransitionRefs>
                                   <TransitionRef Id=”3”/>
                                   <TransitionRef Id=”4”/>
                              </TransitionRefs>
                         </Split>
                    </Transition Restriction>
               </Transition Restrictions>
          </Activity>
          <Activity Id=”4” Name=”D”>
               …
               <Transition Restrictions>
                    <Transition Restriction>
                         <Join Type=”XOR”/>
                    </Transition Restriction>
               </Transition Restrictions>
          </Activity>
     </Activities>
```

```
     <Transitions>
          <Transition Id=”1” From=”1” To=”2” Name=”FromAtoB“/>
          <Transition Id=”2” From=”2” To=”3” Name=”FromBtoC“/>
          <Transition Id=”3” From=”3” To=”2” Name=”FromCtoB“>
               <Condition> … </Condition>
          </Transition>
          <Transition Id=”4” From=”3” To=”4” Name=”FromCtoD“>
               <Condition> … </Condition>
          </Transition>
     </Transitions>
</WorkflowProcess >
```

Реализация регулярного цикла на языке BPML

```
<process name=”RegularCycle”>
     <sequence>
          <action name=”A” …>
          …
          </action>
          <while>
               <condition>
                    …
               </condition>
               <action name=”B” …>
                    …
               </action>
               <action name=”C” …>
                    …
               </action>
          </while>
          <action name=”D” …>
          …
          </action>
     </sequence>
</process>
```

Реализация регулярного цикла на языке BPEL4WS

```
<process name=” RegularCycle”…>
     …
     <sequence>
          <receive … operation=”operation A” …>
               …
          </receive >
          <while condition= … >
               <receive … operation=”operation B” …>
                    …
               </receive >
               <receive … operation=”operation C” …>
                    …
               </receive >
```

```
          </while>
          < receive … operation=”operation D” …>
               …
          </receive >
     </sequence>
</process>
```

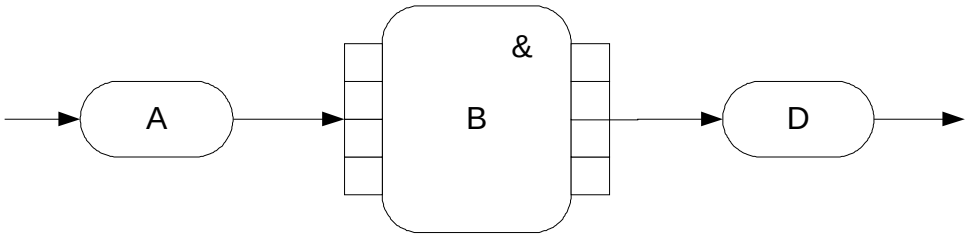
Реализация регулярного цикла на языке jPdl

```
<process-definition name="Cycle">
     …
     <state name="A">
          …
          <transition to="B”/>
     </state>

     <state name="B">
          …
          <transition to="C">
          </transition>
     </state>

     <state name="C">
          …
          <transition to="Decision ">
          </transition>
     </state>

     <decision name="Decision">
          …
          <delegation …>
               …
               if(…) return "continue”;
               else return "stopCycle”;
               …
          </delegation>
          <transition name="continue”  to="B"/>
          <transition name="stopCycle” to="D"/>
     </decision>
     <state name="D">
          …
          <transition to="…">
          </transition>
     </state>
     …
</process-definition>
```

Паттерн "мульти-действие без синхронизации".

Описание паттерна "мульти-действие без синхронизации". Узел в графе workflow-процесса. В момент прихода управления в узел создается несколько экземпляров этого узла, для каждого узла организуется поток управления, и далее в каждом потоке управления параллельно и независимо выполняются разные экземпляры workflow-процесса.

Описание рассматриваемой конструкции: управление находится в узле "A", после того, как выполнено Действие узла A, поток управления переходит в узел "B", в узле "B" создается множество экземпляров узла, которые начинают выполняться, поток управления, не дожидаясь окончания выполнения этих узлов, переходит в узел "D".



Реализация паттерна "мульти-действие без синхронизации" на языке XPDL

В данном случае, паттерн реализуется при помощи запуска в цикле подпроцесса без синхронизации с основным процессом.

```
<WorkflowProcess Id=”1” Name=”MIwithoutSynchronization”>
     …
     <Activities>
          <Activity Id=”1” Name=”A”>
               …
          </Activity>
          <Activity Id=”2” Name=”B”>
               <Implementation>
                    <Subflow … Execution=”ASYNCHR”>
                         …
                    </Subflow>
               </Implementation>
               <Transition Restrictions>
                    <Transition Restriction>
                         <Join Type=”XOR”/>
                    </Transition Restriction>
               </Transition Restrictions>
          </Activity>
          <Activity Id=”3” Name=”C”>
               <Route/>
               <Transition Restrictions>
                    <Transition Restriction>
                         <Split Type=”XOR”>
                              <TransitionRefs>
                                   <TransitionRef Id=”1”/>
```

```
                                        <TransitionRef Id=”2”/>
                                    </TransitionRefs>
                               </Split>
                          </Transition Restriction>
                     </Transition Restrictions>
               </Activity>
               <Activity Id=”4” Name=”D”>
                     …
               </Activity>
          </Activities>
          <Transitions>
               <Transition Id=”1” From=”2” To=”4” Name=”FromAtoB“/>
               <Transition Id=”2” From=”3” To=”4” Name=”FromBtoC“/>
               <Transition Id=”3” From=”1” To=”2” Name=”FromCtoB“>
                     <Condition> … </Condition>
               </Transition>
               <Transition Id=”4” From=”1” To=”3” Name=”FromCtoD“>
                     <Condition> … </Condition>
               </Transition>
          </Transitions>
</WorkflowProcess >
```

Реализация паттерна “мульти-действие без синхронизации” на языке BPML

В этом случае, паттерн также реализуется при помощи запуска в цикле подпроцесса без синхронизации с основным процессом.

```
<process name=” MIwithoutSynchronization”>
     <sequence>
          <context>
               <process name=”B”>
                    …
               </process>
          <action name=”A” …>
          …
          </action>
          <while>
               <condition>
                     …
               </condition>
               <spawn process=”B”/>
          </while>
          <action name=”D” …>
          …
          </action>
     </sequence>
</process>
```

Реализация паттерна “мульти-действие без синхронизации” на языке BPEL4WS

```
<process name=” MIwithoutSynchronization” …>
     …
     <sequence>
          <receive … operation=”operation A” …>
               …
          </receive>
          <while condition= …>
               <invoke … operation=”process B” …>
                    …
               </ invoke >
          </while>
          <receive … operation=”operation D” …>
               …
          </receive>
     </sequence>
</process>
```

Реализация паттерна “мульти-действие без синхронизации” на языке jPdl

```
<process-definition name="Cycle">
     …
     <state name="A">
          …
          <transition to="B”/>
     </state>

     <process-state name="B" process="startNewProcessB">
          …
          <transition to="Decision">
          </transition>
     </state>

     <decision name="Decision">
          …
          <delegation …>
               …
               if(…) return "continue”;
               else return "stopCycle”;
               …
          </delegation>
          <transition name="continue” to="B"/>
          <transition name="stopCycle”  to="D"/>
     </decision>

     <state name="D">
          …
          <transition to="…">
          </transition>
     </state>
     …
</process-definition>
```

Заключение.

Мы привели краткие описания и показали на примерах реализаций некоторых WF-паттернов основные конструкции различных языков.

Несмотря на то, что языков много, они все "распадаются" на два класса принципиально различных языков (граф-ориентированные языки и структурно-ориентированные языки).

У обоих этих классов языков есть серьезные проблемы, например, при помощи граф-ориентированных языков трудно определить такие элементы, как бизнес-исключения, а структурно-ориентированные языки не могут описать бизнес-процессы, соответствующие графам сложной структуры.

В силу несовершенства всех существующих WF-стандартов, как разработчикам софта, так и организациям, выбирающим WF-систему, опасно “жестко” привязываться к какому-то одному из существующих WF-стандартов. Велика вероятность, что в будущем этот стандарт будет кардинально переработан, может быть, даже все современные WF-языки будут вытеснены новым, более удобным, принципиально другим WF-языком.

Таким образом, имеет смысл выбирать WF-систему не "жестко" привязанную к какой-либо спецификации, а гибкую, допускающую импорт-экспорт в различные языки, у которой будет возможность настройки на новые, еще не существующие WF-языки.