

VWL: Veda Workflow Language

Бушенев В.А., Карпов Р.К.
ООО Смысловые Машины
Россия. Республика Коми. Сыктывкар.

Аннотация. После анализа существующих языков моделирования потоков работ, мы остановились на таком замечательном языке как [YAWL](#). Язык YAWL создан на основе сетей [петри](#), однако для преодоления ограничений последних, был обогащен дополнительными механизмами по управлению потоками и преобразованию данных. Созданная спецификация в YAWL представляется в XML формате. YAWL не представлен как самостоятельный язык, а базируется на комплексе включающем в себя компоненты редактора сетей, движка исполнения и хранилища. Такой подход хорош своей комплексностью. Однако, в силу технических ограничений, а также применение онтологий как базового элемента описания и хранения данных в Veda, мы не имели возможности применить это решение для разработанной нами системы. Поэтому, мы пошли путем создания своего луна-парка/языка. Язык VWL внешне напоминает YAWL, заимствует некоторые его идеи и термины, и также имеет графический редактор сетей, среду исполнения и хранения. В этом документе приводится описание языка как в графическом виде с использованием редактора, так и в семантическом виде используя один из форматов представления онтологии.

Описание.

Язык позволяет описать процесс автоматизации последовательности действий агентов. Агентами могут быть как пользователи системы, так и программные скрипты. Реализация данного языка является надстройкой системы Veda, самостоятельное функционирование вне ее не предусмотрено. Созданная на языке VWL спецификация потока работ называется сетью. Сеть представляет собой набор [индивидов](#) хранящихся в системе Veda. С помощью графического редактора доступно визуальное моделирование сети. Созданная сеть может быть исполнена в среде системы Veda, с помощью программой надстройки [Veda Workflow Engine](#).

- * использование данной документации требуется изучения базовых понятий системы Veda.
- * примеры индивидов описываются в формате [Terse RDF Triple Language](#)
- * онтология VWL + VWE описана в <http://semantic-machines.com/veda/veda-workflow>

Veda Workflow Language

v-wf:Net: сеть, состоит из элементов сети, поток выполнения направлен от *v-wf:InputCondition* к *v-wf:OutputCondition*

элементы схемы сети:



v-wf:Condition: узел, может иметь входящие и исходящие потоки



v-wf:InputCondition: стартовый узел с которого начинается выполнение сети



v-wf:OutputCondition: в этом узле заканчивается исполнение сети



v-wf:Flow: поток, имеет направление и может иметь условие для перехода к узлу либо задаче



v-wf:Task: задача, служит для описания каким образом формируются задания для агентов, а так-же описываются условия обработки результатов работы агентов

возможные модификации задачи по управлению потоками:



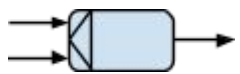
XOR разделение потоков



AND разделение потоков



OR разделение потоков



XOR слияние потоков



AND слияние потоков



OR слияние потоков

структуры описывающие поведение элементов сети:

- *v-wf:VarDefine*: переменная, имеет имя и может иметь указание области видимости, в применении относительно сети может иметь модификатор, входящей, исходящей, либо локальной
- *v-wf:Mapping*: конструкции описывающая как следует заполнять переменную
- *v-wf:ExecutorDefinition*: задает исполнителей для указанной задачи
- *v-wf:Transformation*: структура из набора правил преобразования массива индивидов в новый массив индивидов

VWL в примерах.

В следующих примерах будет описано поэтапное создание сети от простой до сложной.

пример 1. самая простая сеть, ничего полезного не делает.

Создание сети.

главное меню: Документ->Создать, в открывшемся шаблоне, в поле тип набрать "сеть", и выбрать из списка элемент <Сеть>. Далее следует соединить InputCondition и OutputCondition потоком (стрелочка --->). Зададим имя сети в поле [наименование]: "пример сети 1"



Первая наша сеть готова. Для запуска сети потребуется стартовая форма. Документ->Создать, в открывшемся шаблоне, в поле тип набрать "стартовая" и выбрать из списка <Стартовая форма>, в поле [Для сети...] найти нашу сеть "пример сети 1", далее в поле [статус документооборота] внести значение <Ожидает отправки>. После нажатия кнопки сохранить, наша сеть будет запущена на исполнение.

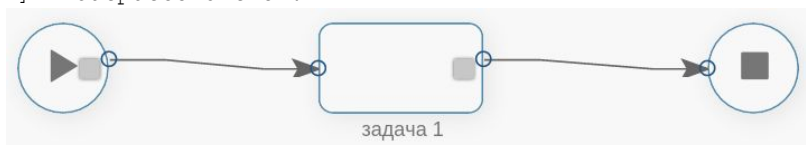
Как проверить результаты исполнения: меню поиск, выбрать из списка <экземпляры маршрута>. Здесь будет список найденных экземпляров запущенных процессов. Найдем наш - "экземпляр маршрута :пример сети 1" и откроем его:



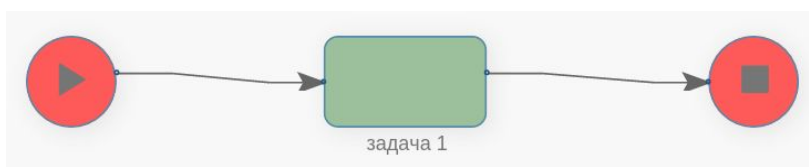
OutputCondition окрашен в красный цвет, это означает что процесс по этой сети был выполнен.

пример 2. сеть с одной пустой задачей, так-же ничего полезного не делает:

Создадим новую сеть аналогично примеру 1, однако в новую сеть мы добавим задачу. Так же в новой сети будет два потока: из InputCondition в [задача 1] и из [задача 1] в OutputCondition.



Запустим сеть аналогично примеру 1:



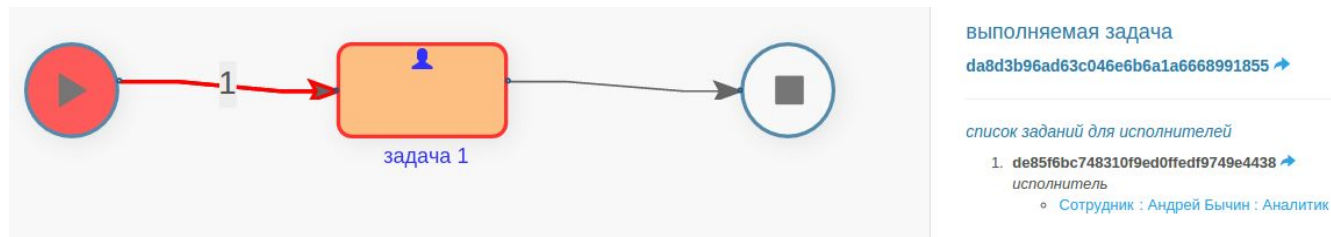
Видно что исполненная задача отмечена зеленым цветом. Если выделить какой либо из элементов сети, то в правой панели будет подробная информация о состоянии элемента сети относительно его исполнения.

пример 3. сеть которая в которой исполнителем указан сотрудник

Создадим новую сеть аналогично примеру 2, укажем в качестве [исполнителя] конкретного человека: Андрей Бычин: Аналитик



Запустим сеть :



Видно что выполнение в отличие от примера 2, остановилось на задаче 1. Однако если мы поищем задачи, которые должны прийти сотруднику Андрею, там ничего не будет. Это связано с тем что движок workflow довольно абстрактен и ничего не знает о том как должны выглядеть формы задач (v-wf:DecisionForm) на которые должен отвечать пользователь. А пустую форму движок пока не умеет создавать. В данной ситуации можно и вручную продвинуть исполнение сети, но это потребует детальных технических знаний о внутренностях движка. Что будет описано ниже в разделе "Veda Workflow Engine, как это работает."

пример 4. сеть которая выдает задание сотруднику

Для того чтобы создать форму ответа на задачу, нам потребуется структура v-wf:Transform. С помощью v-wf:Transform мы зададим правила трансформации которые сформируют для нас пользовательскую форму ответа на задачу.

Загрузим в систему следующий фрагмент онтологии:

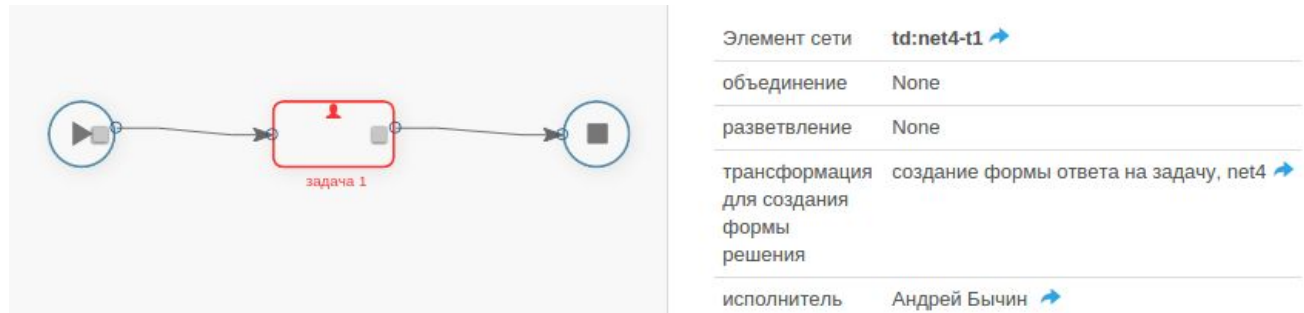
```
:net4-tr1
  rdf:type v-wf:Transform ;
  rdfs:label "создание формы ответа на задачу, net4"^^xsd:string;
  v-wf:transformRule :net4-tr1-r1 ;
.

:net4-tr1-r1
  rdf:type v-wf:Rule ;
  v-wf:segregateElement "contentName('@') " ;
  v-wf:aggregate "putUri ('rdf:type', 'v-wf:DecisionForm')";
  v-wf:aggregate "putUri ('rdf:type', 'mnd-wf:UserTaskForm')";
  v-wf:aggregate "putString ('rdfs:label', 'задание')";
  v-wf:aggregate "putBoolean ('v-wf:isCompleted', false)";
  v-wf:aggregate "putExecutor ('v-wf:to')";
  v-wf:aggregate "putWorkOrder ('v-wf:onWorkOrder')";
  v-wf:aggregate "putUri ('v-wf:possibleDecisionClass', 'v-wf:DecisionAchieved')";
  v-wf:aggregate "putUri ('v-wf:possibleDecisionClass', 'v-wf:DecisionNotPerformed')";
.
```

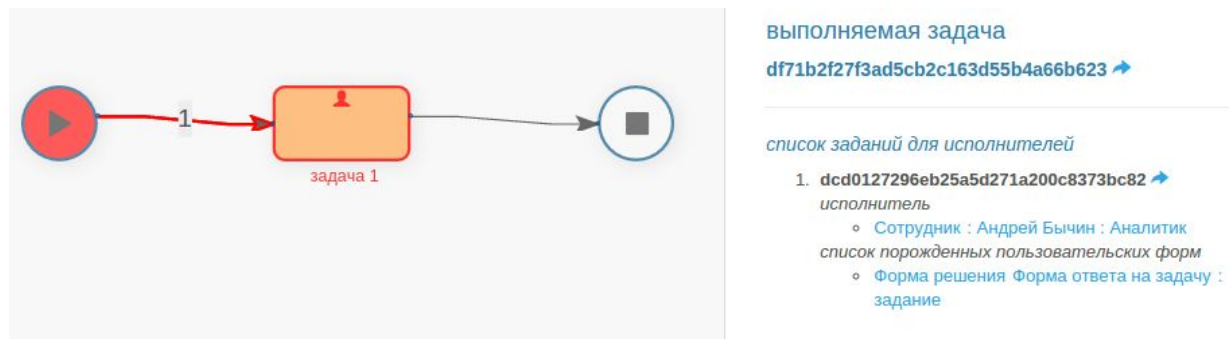
Как этот пример преобразования работает: индивид типа v-wf:Transform, содержит одно или более правил преобразования данных v-wf:Rule. Когда подойдет время выполнения элемента задача 1. Будет проверено поле [трансформация для создания формы решения], и при наличии правил преобразования данных, выполнится указанная трансформация. На вход трансформации будет подан массив переменных исполняемой задачи. Так как мы сами не задавали никаких переменных в сети или задаче, массив будет содержать только одну переменную curTask, которую создает сам движок workflow.

Поле [v-wf:segregateElement] содержит js выражение, который отфильтрует только одно поле '@' из всех полей содержащихся в переменной curTask. Далее будут выполнены все выражения из поля [v-wf:aggregate]. Результатом их исполнения будет индивид типа v-wf:DecisionForm, так-же движком будут выданы права для исполнителя на изменение этого индивида. Более подробно о преобразовании данных в разделе [Трансформация: как это работает](#).

и ссылку на :net4-tr1, впишем в элемент сети задача 1, в поле [трансформация для создания формы решения]



запустим сеть :



Видно что в правой информационной панели появилась ссылка на созданную пользовательскую форму.

Эту форму можно найти во входящих задачах сотрудника Андрея. Выглядит она так:

Форма ответа на задачу [d9fe823779dad370496650246e1b4b93](#)

задание

от кого кому когда
Андрей Бычин : Аналитик

решение

выполнено
выполнено
не выполнено

по документу

Отправить

Сразу бросается в глаза что в задаче не указано от кого она пришла и что собственно нужно сделать, в следующем примере добавим это.

пример 5. сеть которая выдает задание сотруднику с указанием от кого и что следует сделать.

Если в предыдущих примерах, для нашей сети не требовались входные параметры, то в этом примере зададим два входных параметра, первый – кто запустил сеть и сообщение о том что нужно сделать.

Для этого нам понадобится такая сущность, как переменная.

`v-wf:Variable` это переменная, которая имеет имя, значение и область видимости. Переменные можно задать как для сети, так и для каждой из задач.

Описываются переменные конструкцией `v-wf:VarDefine`, это описание безотносительно сети или какого либо ее элемента.

Создадим переменную *initiator*.

Трансформация: как это работает.

Алгоритм работы трансформации простой: вычленим из входной информации только нужное нам, преобразуем отобранное, и последним этапом группируем.

Рассмотрим подробнее этот процесс. На вход функции трансформации подается массив индивидов, и ссылка на правила преобразования. На выходе будет новый массив индивидов. При этом размер выходного массива может быть как меньше входного так и больше, либо не меняться. Другими словами, к примеру, мы можем сделать из одного индивида несколько, либо наоборот из нескольких одного.

Фазы преобразования данных:

- A. фильтрация информации на уровне индивидов и их полей
- B. преобразование полей и сохранение новых во временный буфер
- C. группировка полей
- D. создание новых индивидов

Первые две фазы исполняются столько раз, сколько у нас есть полей во всех индивидах, иначе говоря алгоритм обходит все поля всех исходных индивидов используя два вложенных друг в друга цикла. Далее текущий обрабатываемый в цикле индивид будем называть **объект**, а текущее поле во сложном - **элемент**.

Класс трансформации `v-wf:Transform` содержит в себе ссылки на правила преобразования `v-wf:Rule` которые в свою очередь описывают выражения для фаз преобразования -

- `v-wf:segregateObject` - фильтрация индивидов
- `v-wf:segregateElement` - фильтрация полей
- `v-wf:aggregate` - преобразование полей
- `v-wf:grouping` - группировка

Доступные js функции по фазам:

A. фильтрация по индивидам: `v-wf:segregateObject` `[expression == true/false]`

- `objectContentStrValue (name, value)` - объект содержит поле `[name]` с содержимым `[value]`

A. фильтрация по полям: `v-wf:segregateElement` `[expression == true/false]`

- `contentName (name)` - элемент имеет имя `[name]`
- `elementContentStrValue (name, value)` - элемент имеет имя `[name]` и содержит строковое значение `[value]`

B. преобразование: `v-wf:aggregate` `[expression == {data: xxx; type: ttt}]`

- `getElement ()` - возвращает значение элемента
- `putFieldOfIndividFromElement (name, field)` - сохраняет в буфер поле с именем `[name]` взятое из поля `[field]` индивида найденного в базе по ссылке содержащейся в значении элемента
- `putElement (name)` - сохраняет в буфер поле с именем `[name]` и значением из элемента
- `putFieldOfObject (name, field)` - сохраняет в буфер поле с именем `[name]` взятое из поля `[field]` объекта
- `putUri (name, value)` - сохраняет в буфер поле с именем `[name]` и значением `[value]` и типом `Uri`
- `putString (name, value)` - сохраняет в буфер поле с именем `[name]` и значением `[value]` и типом `String`
- `putBoolean (name, value)` - сохраняет в буфер поле с именем `[name]` и значением `[value]` и типом `Boolean`
- `putExecutor (name)` - сохраняет в буфер поле с именем `[name]` ссылки на исполнителей сети
- `putWorkOrder (name)` - сохраняет в буфер поле с именем `[name]` ссылки на рабочее задание

Пример 1: преобразование - много -> один

допустим у нас есть массив из нескольких индивидов:

```
tst:individual1
  rdf:type tst:colorA ;
  rdfs:label "red" ;
  v-s:login "Bychina" .

tst:individual2
  rdf:type tst:colorB ;
  rdfs:label "green" ;
  v-s:login "KarpovR" .

tst:individual3
  rdf:type tst:colorA ;
  v-s:login "KarpovR" .

tst:individual4
  rdf:type tst:typeA ;
  rdfs:label "long" .
```

а нам нужен один индивид (xxxx - любой id), содержащий некоторые поля из нескольких индивидов:

```
:xxx1
  rdf:type :typeX ;
  tst:colorA "red" ;
  tst:colorB "green" ;
  tst:typeA "long" .
```

Для начала отфильтруем нужные нам индивиды:

```
v-wf:segregateObject "objectContentStrValue ('rdf:type', 'colorB') ||
objectContentStrValue ('rdf:type', 'colorA')"
```

Далее отфильтруем нужные поля: `v-wf:segregateElement "contentName('rdf:type')"`

Теперь преобразуем данные: `v-wf:aggregate "putFieldOfObject (getElement(), 'rdfs:label')"`

А так-же добавим тип для нового индивида: `v-wf:aggregate "putUri ('rdf:type', ':typeX')";`

итого получилось правило:

```
tst:transformation1
  rdf:type v-wf:Transform ;
  v-wf:transformRule tst:rule1, tst:rule2.

tst:rule1
  rdf:type v-wf:Rule ;
  v-wf:segregateObject "objectContentStrValue ('rdf:type', 'colorB') ||
objectContentStrValue ('rdf:type', 'colorA')" ;
  v-wf:aggregate "putFieldOfObject (getElement(), 'rdfs:label')";
  v-wf:grouping "1".

tst:rule2
  rdf:type v-wf:Rule ;
  v-wf:segregateObject "objectContentStrValue ('rdf:type', 'colorB') ||
objectContentStrValue ('rdf:type', 'colorA')" ;
  v-wf:aggregate "putUri ('rdf:type', ':typeX')";
  v-wf:grouping "1".
```

Пример 2: преобразование - один -> много

дано:

```
tst:indivdX
  rdf:type :typeX ;
  tst:color "red" ;
  tst:color "green" ;
  tst:color "blue" .
```

требуется получить:

```
:xxxx1
  rdf:type :typeY ;
  tst:color "red" .

:xxxx2
  rdf:type :typeY ;
  tst:color "green" .

:xxx3
  rdf:type :typeY ;
  tst:color "blue" .
```

В этом примере нам не понадобится `v-wf:segregateObject` так как фильтровать объекты не нужно, входящий массив содержит один индивид.

правило:

```
tst:transformation2
  rdf:type v-wf:Transform ;
  v-wf:transformRule tst:rule21 .

tst:rule21
  rdf:type v-wf:Rule ;
  v-wf:segregateElement "contentName('tst:color')";
  v-wf:aggregate "putUri ('rdf:type', ':typeY')";
  v-wf:aggregate "putElement ('tst:color')".
```

VWE: Veda Workflow Engine

Бушенев В. А. Максименко В. В.
ООО Смысловые Машины
Россия. Республика Коми. Сыктывкар.

Аннотация.

почему был выбран для движка принцип триггерный обработки

Ограничения данной реализации:

- OR и XOR слияния потоков не реализованы.
- Агентами являются v-s:Appointment и v-s:Codelet

Veda Workflow Engine, как это работает.

объекты VWE:

- v-wf:StartForm: стартовая форма с исходными данными процесса
- v-wf:Process: процесс, описывает запущенный экземпляр сети net
- v-wf:WorkItem: рабочий элемент, описывает запущенный экземпляр задачи task
- v-wf:WorkOrder: рабочее задание для конкретного исполнителя
- v-wf:DecisionForm: форма ответа(решения) пользователя
- v-wf:Variable: экземпляр переменной

принцип работы VWE:

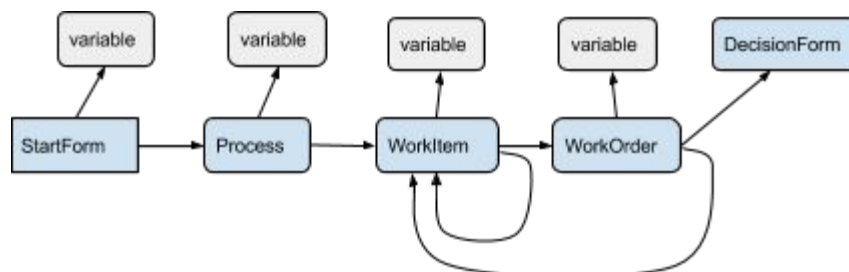
Онтологии и javascript работающий на стороне сервера, позволяющий исполнять схемы описанные на VWL

В основе работы VWE лежит - такое свойство системы veda, как возможность исполнения заданного javascript по событию создания либо изменения индивида, это аналог триггеров в SQL базах данных.

Таким образом обработка сети начинается с индивида типа v-wf:StartForm, которая порождает v-wf:Process, который в свою очередь находит в сети v-wf:InputCondition и порождает для него WorkItem. Последний порождает в зависимости от ситуации либо снова v-wf:WorkItem, либо v-wf:WorkOrder, которые обрабатывают агенты, и помещают в них результаты своей деятельности. Обработка v-wf:WorkOrder порождает v-wf:WorkItem. Заканчивается выполнение сети обработкой элемента сети типа v-wf:OutputCondition, в процессе обработки этого элемента, ни каких новых v-wf:WorkItem не порождается. А значит далее нечего выполнять касательно данной сети.

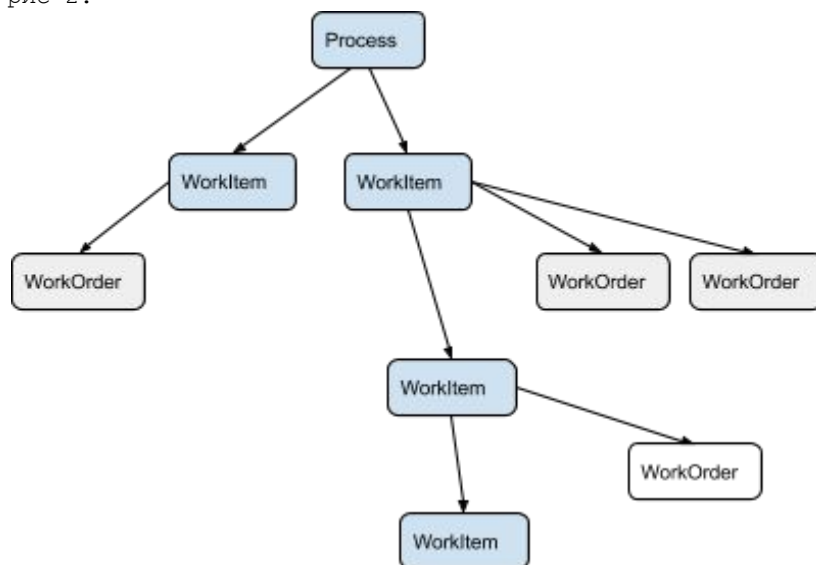
Схематично процесс порождения объектов можно представить в виде диаграммы.

рис 1.



все созданные индивиды в процессе обработки связываются друг с другом, таким образом создается дерево выполнения сети, которое можно обойти для дальнейшего анализа.

рис 2.



Теперь подробнее о каждом этапе исполнения сети.

1. старт сети

исполнение сети начинается с обработки события создания индивида типа `v-wf:StartForm`, это стартовая форма запуска процесса, она обязана содержать следующие поля:

`v-s:hasStatusWorkflow = v-s:ToBeSent`

`v-wf:forNet` - указание, какая сеть должна быть запущена

`v-wf:useTransformation` - указание на правила преобразования данных, подготавливающие переменные сети, обычно они формируются из других полей данной стартовой формы.

В процессе обработки стартовой формы будет создан процесс из `v-wf:forNet`, а так-же необходимые переменные, согласно правилам преобразования (`v-wf:useTransformation`). Переменные будут привязаны к процессу в поле `v-wf:inVars`. Обработка стартовой формы закончена.

2. обработка экземпляра `v-wf:Process`

Первым шагом обработки создаются локальные переменные для текущего процесса, затем в сети которую представляет процесс, происходит поиск элемента типа `v-wf:InputCondition`. Для такого элемента подготавливается экземпляр `v-wf:WorkItem`, который связан с обрабатываемым процессом и его сетью, а так-же привязан к найденному элементу типа `v-wf:InputCondition`.

Вновь созданный экземпляр `v-wf:WorkItem` привязывается к текущему процессу через поле `v-wf:workItemList`. Экземпляр `v-wf:WorkItem` сохраняется в базе данных. Обработка процесса закончена.

3. обработка экземпляра `v-wf:WorkItem`

если поле `v-wf:isCompleted == true`, то обработка прекращается.

рассматривается поле `v-wf:join`, если `== v-wf:AND`, Так как каждая из входящих задач порождает `v-wf:WorkItem` для текущей задачи, то данная проверка на `v-wf:join == v-wf:AND`, будет происходить каждый раз. Далее найдем все `v-wf:WorkItem` порожденных от задач имеющих выходы к текущей задаче и проверим все ли они были успешно завершены, если да, то продолжим обработку, иначе закончим обработку.

! тут нужно обратить внимание, что из всех входов от других задач, пройдет дальше только один из них, первый из списка, и для отслеживания по дереву выполнения сети нужно будет один из путей.

Далее, если тип элемента == v-wf:Task выполняется формирование входящих переменных задачи из поля v-wf:startingMapping.

здесь происходит вычисление исполнителей (агентов) для задачи, данные для вычисления берутся из поля v-wf:executor. Здесь могут быть индивиды трех типов: v-s:Appointment, v-wf:Codelet, v-wf:executorExpression. Последний представляет собой javascript выражение результатом работы которого будет список из индивидов типа v-s:Appointment или v-wf:Codelet.

По по каждому элементу из списка исполнителей будут порождены задания: экземпляры типа v-wf:WorkOrder. В каждом из них будет указание на текущий рабочий элемент, на исполнителя, а так-же если задача содержит указание на под-сеть, то она будет указана в рабочем задании.

Если не было найдено ни одного исполнителя, то будет сформировано пустое рабочее задание.

Список сформированных рабочих заданий будет включен в текущий рабочий элемент в поле v-wf:workOrderList, в дальнейшем этот список будет использоваться для определения, все ли рабочие задания были исполнены.

На этом обработка завершается.

Если тип элемента == v-wf:InputCondition
происходит выбор следующих элементов из полей v-wf:hasFlow->v-wf:flowsInto и порождение для каждого из них соответствующих рабочих элементов.

Если тип элемента == v-wf:OutputCondition

если был указан v-wf:parentWorkOrder то выполняется формирование переменных по условиям из поля v-wf:completedMapping которые сохраняются в [v-wf:parentWorkOrder]->v-wf:outVars. В случае отсутствия условий для формирования переменных в v-wf:outVars помещается v-wf:complete. Далее в рабочий элемент помещается поле v-wf:isCompleted = true.

4. обработка экземпляра v-wf:WorkOrder

[Обработка новых рабочих заданий]

Здесь берутся только необработанные рабочие задания.

если не указаны исполнители, то в исходящие переменные (v-wf:outVars) заносится v-wf:complete.

если тип исполнителя v-s:Codelet, то проводим прямое исполнение заданного скрипта и тут же обрабатываем результаты его работы, путем преобразования результатов в переменные с помощью указаний в v-wf:completedMapping и помещения их в v-wf:outVars далее переходим к [Обработка результатов рабочих заданий].

если тип исполнителя v-s:Appointment, то производим формирование DecisionForm для пользователя, с помощью правил трансформации указанных v-wf:startDecisionTransform. Так же выдаются права исполнителям на редактирование вновь созданных DecisionForm

если исполнитель v-wf:Net или указано что используется подсеть (v-wf:useSubNet == true), то генерируем новый под-процесс.

Обработка завершена.

[Обработка результатов рабочих заданий]

5. обработка экземпляра DecisionForm

если `v-wf:isCompleted == true` или поле `v-wf:takenDecision` не заполнено, то
обработка завершается