Workflow: A powerful workflow engine

Cédrik Béler, Stéphane Ducasse and Max Leske

December 2, 2017
master @ ec63f10*

Layout and typography based on the `sbabook` LaTeX class by Damien Pollet.

Contents

Illustrations

CHAPTER 1

Introduction

The document is about the (re)design of the Pharo package Workflow, a powerful and extensible workflow management system.

Its initial name was Aare and it was part of a proprietary application entirely web-based that was successfully integrated with a document management system, enabling one to attach metadata annotated documents to workflows and activities. One special component of Aare was the feature-rich workflow engine allowing one to design complex business workflows, run and control them, collect necessary data through sophisticated electronic forms and automatically notify responsible people.

It was recently open-sourced by Netstyle[1]. What's missing compared to the full (proprietary) solution are user interfaces that were web based (Seaside + Magritte). Also, persistence that was achived through Omnibase. References to Omnibase were removed but they are stil some disfunctionning. Moreover, the completion of activities were mainly manual and achieved through web interface (web forms).

For now, WfWorkflow is quite usable and quite well tested. It is possible to create processes, activities, transitions and run them (actually complete them) through an simple engine.

Instead of mimicking what is missing, the objective is to develop a generic business workflow engine for Pharo. It has to be loosely coupled with the persistence solution and user interactions solution.

This document first objective is to present the current WfWorkflow design. Most of the information will be extracted from the white paper and by orginal

[1] http://www.netstyle.ch

code comments, plus some discussions with several people of the Pharo communauty and especially Max Leske who is from Netstyle[2].

The second objective is to deal with important dessign decision to ensure WfWorkflow being a powerful and extensible workflow management system for Pharo. There are 2 main tasks:

rethink persistence

(process definition and orchestration, realization) Persistance is essential and was central to Aare. Beside removing Omnibase references (and implication of WfManagedObject), some of the application features were very dependant on the persistence (like logging, tracing process evolution). All of these features are to be thing again considering as much as possible a loosely coupled solution. A general purpose storage solution like Voyage could be used. At first, in image storage will be used to focus on the running aspects of the library.

rethink interaction

Designing a general interaction sub-system is probably the more challenging tasks. Aare was web based only although based on Magritte that could possibl enable morphic rendering. We want workflow to be UI agnostic. Even more, completion of activities are achieved through activation management (a special class of WFWorkfkow that is central and that need more love). One can imagine more than user interaction with automated interactions through service for instance.

> **To do** Activation should be managed through several possibilities being message-based, event-based, service-based. Ideally there should be hook to implement proper interfaces. Annoucement seems a good condidate to build an independant interaction system.

The booklet is clearly a work-in-progress and will evolve according to discussions and code submissions.

The booklet is organized around several parts:

- Part I - Workflow by Example Quick way (and current way) of using the business process engine.
- Part II - Core design and key classes
- Part III - Persistance discussions
- Part IV - User and automatic interaction (UAI)
- APPENDIX (code comments, ...)

[2] http://www.netstyle.ch

Lots of information were found in Netstyle whitepaper[3] about Aare.

[3] https://github.com/Netstyle/Workflow/blob/master/Workflow-whitepaper.pdf

CHAPTER 2

Hands on WfWorkflow

In this chapter, a quick overview of what is possible with WfWorkflow is shown and also the limitations of the package. The objective is to give the big picture of the API, of the different components interacting.

For a more complete explanation of the design, see Chapter **??**.

Some design discussions will be exposed in this chapter, especially concerning the missing parts of the package. This discussions will be developed in detail in Chapter **??**.

The following explains how to use WfWorkflow.

2.1 WfWorkflow installation

Tested on Pharo 6.1.

On github

Workflow is hosted on the following github repository: `https://github.com/Netstyle/W`

Evaluate the following in a workspace to quickly load the project:

There are errors with the actual version.

Listing 2-1 "Do it" in a workspace.

```
Metacello new
    baseline: #Workflow;
    repository: 'github://Netstyle/Workflow:master/src';
    load.
```

```
*** Warning: Warning: This package depends on the following classes:
  WADynamicVariable
You must resolve these dependencies before you will be able to load
    these definitions:
  WfCurrentWorkflowManager

WfTestArchiving>>assertRoundtripFor: (DomBuilder is Undeclared)
WfTestArchiving>>assertRoundtripFor: (IDBasicManager is Undeclared)
WfTestArchiving>>assertRoundtripFor: (IDCurrentManager is Undeclared)
WfTestXPDL>>assertRoundtripFor: (DomBuilder is Undeclared)
WfTestXPDL>>assertRoundtripFor: (IDBasicManager is Undeclared)
WfTestXPDL>>assertRoundtripFor: (IDCurrentManager is Undeclared)
WfImporter>>import (IDCurrentManager is Undeclared)
WfRootFrame>>initializeWithWorkflow: (TimeStamp is Undeclared)
WfManagedObject>>workflowManager (WfCurrentWorkflowManager is
    Undeclared)
```

To do Solve these errors (Workflow-cypress.1)

- a temporary WADynamicVariable is created, subclass of DynamicVariable
- DomBuilder need to be replaced by XMLParser.
- IDBasicManager IDCurrentManager ?
- TimeStamp reference is replaced by an equivalent.

2.2 Package overview

The package name is `=Workflow=`. Its goal is to specify, execute, monitor workflow realization. It's all about coordinating the flow of information. One important objective it to do this coordination in a distributed environment.

Workflow sub-package composed of several sub-packages named respectively workflow, archiving, conditions, Tests, XPDL, XPDL-import and XPDL-export. The workflow sub-packagecontains most of the core classes, whether for the static definition of a workflow or for the workflow engine. conditions sub-package contains also workflow specification classes but only used to define transition conditions. Archiving is for the monitoring

2.3 Main classes

The following is a quick description of the most important classes that allows to specify a workflow (1) and execute it (2). Monitoring is not yet functionning (3)

`WfManagedObject` inherits of `Object` was originally subclass of Omnibase managed objects (persistance). Now, there are no persistance by default. One

option would be to use voyage and adapt the code to provide persistance on different backends. Lots of classes inherits from `WfManagedObject`.

WfManagedObject comment: I provide common functionality for peristent objects within the workflow engine.

Workflow definition (1)

WfStaticElement and its subclasses are responsible of workflow specification/definition. It is said static as it can be seen as a model of future succession of activities. Here are a quick description of the main classes.

WfWorkflow is an planified operational aspect of a work procedure. A workflow knows :

- how activities are structured
- who perform activities
- the relative order of activities
- how activities are synchronized
- how information flows to support activities (electric form mainly in Aare)
- how workflows are tracked (evolution records)

WfWorkflow comment: I am the main entity of a workflow definition. I know about my all my steps, including the start steps and the history of myself. For opimization I am caching the looping edges within a dictionary.

Activities are represented by `WfStep` subclasses. There are different kinds of `WfSteps` (activity, named, start, substeps that are static or dynamic). `WfNamedStep` is a simple activity. `WfStep` subclasses represent activity parts of a workflow to do a certain job and they define:

- form definitions to collect data
- form conditions to validate entered data
- transitions to subsequent activities
- transition conditions to decide which activity to trigger on completion
- other settings: responsible users, expected duration, documents, ...

WfStep comment: I am an abstract step (activation) to be used as the nodes of a workflow definition. I have got references to the outgoing edges and the workflow I am used in.

WfStartStep comment (WfStep subclass): I am a special step to be used as a starting point of the workflow. Every workflow has got exactly one start-step.

WfNamedStep comment (WfStep subclass): I am the basic workflow step that is used in most cases having a name and an additonal field for comments.

WfSubflowStep comment (WfStep subclass): I am an abstract step (activation) to be used as the nodes of a workflow definition. I have got references to the outgoing edges and the workflow I am used in.

WfStaticSubflowStep comment (WfSubflowStep subclass): I am an abstract step (activation) to be used as the nodes of a workflow definition. I have got references to the outgoing edges and the workflow I am used in.

WfDynamicSubflowStep comment (WfSubflowStep subclass): I am an abstract step (activation) to be used as the nodes of a workflow definition. I have got references to the outgoing edges and the workflow I am used in.

Transitions between activities are represented by **WfOutgoingEdge** (transitions form a steps to another).

WfOutgoingEdge comment: I am an outgoing edge to be used in the workflow-definition. I have got references to the source and destination step, as well as a condition that defines if this edge can be used when running the workflow.

On outgoingEdges, conditions can be affected. There are several kind of them, all present in the condition sub-package.

WfCondition comment: I am an abstract condition.

WfTrueCondition comment: I am a condition that is always satisified.

WfFalseCondition comment: I am a condition that can never be satisfied.

WfValueCondition comment: I am a condition that is satisfied if the checked object has a property named key (#propertyAt:) that is equal (#=) to value.

WfComposedCondition comment (abstract): I am an abstract composed condition referencing other conditions.

WfAllCondition comment (WfComposedCondition subclass): I am a condition that is satisfied if ALL of my children are satisfied. It is a generalized AND relation.

WfAnyCondition comment (WfComposedCondition subclass): I am a condition that is satisfied if ANY of my children are satisfied. It is a generalized OR relation.

Considering all this classes, the workflow package supports several workflow patterns:

- sequential routing: activation of B is enabled once activation A is completed
- conditionnal routing: control point according to entered data (or a computed value ?).

- parallel routing: multiple thread of control (like the simple XOR with 2 true condition on the outgoing edges)
- syncmerge
- loop multiple instance without synchronized
- multi instance with a priori runtime knowledge
- sub-workflow routing (definition - static)
- sub-workflow routing (runtime - dynamic)
- implicit termination (no more active activations and no more possible)

Workflow execution (2)

Once workflow are specified, they are executed on demand (eventually scheduled). Workflow specification are like a plan and its realization has to conform to it (the more possible - real execution has to face problems). Once a workflow is executed, the main role of the worflow engine is to coordinate all activities that compose the worflow. Depending on transition and upon activity completion, the engine exposes the new tasks/activities that need to be done. Some other important goal of the engine is to monitor the workflow execution (see the next section).

In Workflow parlance, once a workflow is executed, its lone start activity (instance of `WfStartStep`) is *activated*. It transparently activates the next step(s). According to transition conditions, once a step is completed, then, the next available steps are activated. `WfActivation` are the class that is instanciated each time a `WfStep` is actually activated. A running activation corresponds to an activity that is running (eventually suspended, idled).

WfActivation represents the realization of an activity as part of a running workflow. It's a central classe of the workflow engine. It has no subclass at this stage. Since there are split or loops in a given workflow, multiple activations of different or the same activity can exist. Moreover , if several workflow are executed in the same time, multiple activations can be active at the same time. On the actual Workflow version, there are no WfActivation subclass (there should be and there a trace of previous BLActivation used in the original Aare application).

WfActivation comment: I am an activation to be used in a running-workflow. The activation has got references to its step (activity) and its frame. What is more it consists of a dictionary of properties that are used to store domain-data. Most extension methods of this class should be pushed down into BLActivation.

So as to keep track and manage workflow real evolution, a `WfFrame` is used. **`WfFrame`** represent the frame to be used in a running-workflow.it has got a reference to the workflow and it is based on a dictionary mapping steps to

an activation. It exists several subclasses (WfRootFrame for the main process frame, SubFrame to embed sub-workflow, ...).

WfFrame comment: I am a frame to be used in a running-workflow. I have got a reference to the workflow I am based on and a dictionary mapping steps to an activation.

WfRootFrame represents the main frame of a workflow execution.

WfRootFrame comment: I am the root frame of a running workflow, the main entity of the workflow execution engine. I reference a work-list with all the completed and running activations. I also contain some domain-data including properties such as the title of the running workflow, the timestamp when the workflow has been started, a priortity and the user that started the workflow.

There are also dedicated sub-frame to deal with static and dynamic subflows.

WfSubFrame comment: I am an abstract sub-frame having a link to the parent frame.

WfLoopingFrame comment (WfSubFrame subclass): I am a looping frame, to be used when the edges of a workflow are looping.

WfSubflowFrame comment (WfSubFrame subclass): I am the frame of a subflow, I have got my own worklist to keep track of activations within that subflow and a reference to the step that started myself.

WfWorkList represents the worklist of a running-workflow. It has two collections, one with the running and one with the completed activations. It is a major components of the engine. For now it has no subclasses. It is a kind of simple scheduler or at least activation queue.

WfWorklList comment: I am the worklist of a running-workflow. I have two collections, one with the running and one with the completed acitvations.

WfWorkflowHistory represents the versions of a workflow definition. He has got a name and references to all the versions of my workflows. He is able to create new versions of a workflow by copying the latest one.

WfWorkflowHistory comment: I represent the versions of a workflow definition. I've got a name and references to all the versions of my workflows. What is more I am able to create new versions of a workflow by copying the latest one.

WfWorkflowManager role it to manage the changes within definitions and running workflows. Right now it has no subclasses and define only the protocol (ex: noteRemovalOfEdge:fromStep:)

WfWorkflowManagery comment: I manage the changes within definitions and running workflows.

Other classes

WfWorkflowLibrary that is a collection of some simple workflow definitions (simpleSequence simpleSplit simpleBranch simpleJoin simpleLoop simpleBranchJoin branchAndLoop loopingJoin simpleSubflow loopingSubflow dynamicSubflow)

In the archiving package (3), class dedicated to the storage of process executions throught activation log/records are defined. Yet, there are not working right now as they rely on two missing components (DomBuilder, IDBasicManager, IDCurrentManager and IDBasicManag).

> **To do** right now, code exists (class ending with exporter) but they do not work a We have to replace it or ask Max for these components.

In the tests package, there are several unit tests. With small modifications (dumb loading problem solutions), the core tests are green (concern the specification of several kind of workflow and their execution in some ways mainly be sending manually complete to running activations). All hitory, archiving, or importing/exporting are failling as expected since the Workflow package is deconnected from the closed source application. Nevertheless their protocol gives indication on how such functionnlities behave so the will be used as blueprints to the new implementation.

1.

In xPDL xPDL-Export xPDL-Import packages, there are code allowing the serialization of the workflow in the xPDL standard. Code is availbale but an XMLParser is missing right now. Either Netstyle will provide some of thises classes or we will adapt the stand pharo XMLParser.

2.4 Code snippets

As I discovered the package and how it works, here are some code snippets to use the package.

Declaring a workflow

Several exemples of simple workflow definition can be found in the `WfWorkflowLibrary` class. For instance, here is a definition of a simple branch workflow.

The workflow representation is:

```
                        (yes) branch1
Start => branchPoint =>
                        (no)  branch2
```

Below are two way to version used to model such a workflow.

```
    WfWorkflowLibrary>>>simpleBranch
  | workflow branch1 branch2 branchPoint |
  workflow :=  self workflowNamed: 'Simple XOR Branch'.
  branch1 := self newStepNamed: 'Branch 1' in: workflow.
  branch2 := self newStepNamed: 'Branch 2' in: workflow.
  branchPoint := self newBranchPointNamed: 'Brancher' for: branch1
    and: branch2 in: workflow.
  workflow start addOutgoingEdgeFor: branchPoint.
  ^ workflow
```

On this library, several helpers functions defined and there used in the example above. Without any helpers, here is how the code look like to declare the same workflow:

```
| workflow history branch1 branch2 branchPoint|
workflow := WfWorkflow new. "reset steps (empty Set) and create the
    start step"
"To add to the engine history: (not working)"
history := WfWorkflowHistory forWorkflow: workflow.
history name: 'Simple XOR Branch'. "name of a workflow is through
    its history only ?"
"a workflowmanager sublclass os DynamicVaribale is also concerned"
branch1 := workflow newStepNamed: 'Branch 1'.
branch2 := workflow newStepNamed: 'Branch 2'.
workflow addSteps: (Array with: branch1 with: branch2).

branchPoint:= workflow newStepNamed: 'Brancher'.
branchPoint addOutgoingEdgeFor: branch1.
branchPoint addOutgoingEdgeFor: branch2.
branchPoint atOutgoingEdgeFor: branch1 putCondition:
    (WfValueCondition property: #branch value: true).
branchPoint atOutgoingEdgeFor: branch2 putCondition:
    (WfValueCondition property: #branch value: false).
workflow addStep: branchPoint.

workflow start addOutgoingEdgeFor: branchPoint.
```

WfWorkflow»>addStep is used through convenience methods like:

- WfWorkflow»>newStartStep
- WfWorkflow»>newNamedStep
- WfWorkflow»>newActivityStep (same implementation as newNamedStep)
- WfWorkflow»>newStartStep
- WfWorkflow»>newDynamicSubflowStep

But they just create activities (steps) that need to be later associated to outgoing edges + conditions. (#addOutgoingEdgeFor: and atOutgoingEdgeFor:putCondition:).

To do create convenience method to build a workflow, like in WfWorkflowLibrary ?

2.5 Execution of a workflow

Execution in the initial version seems limited to two kind of activation. Most of the time, UI was sending a completion message to a current activation. We explain it below.

Running a workflow

Once a workflow is defined, to be executed, he calls the method #executeInNewFrame. This method instanciate and populate the worklist (WfWorklist) that keep track of the state of the running workflow.

```
workflow := WfWorkflowLibrary simpleBranch.
frame := workflow executeInNewFrame.
```

To see running activitions (activities that are started but not completed yet):

```
frame worklist running.
```

Once a activation is over, it has to be completed by sending the message #completed. For now, there are only manual completions but we could imagine in the future automatic or semi-atomatic completion based on extrenal interaction or if delegated to a program for instance.

```
anActivation := frame worklist any.
anActivation complete. "complete mannully one activation"
"or"
frame complete: anActivation.
```

The workflow is running until there are no possible (accessible) activation left. This state is called implicit termination and can be checked with:

```
frame isCompleted.
```

Here is the code of the WfFrame»>completeAll

```
WfFrame>>>completeAll
  [self workList allRunning isEmpty] whileFalse:
    [self completeActivation: self workList allRunning first].
```

Important methods

Dynamic workflow resolution

It seems the orinal goal of the workflow engine was to consider a workflow definition frozen once it's started to be run. If the process model is modified, only new instances will be concerned.

This restriction is not enforced by design and eventually that could be far more interesting to consider process executions as activations based on the model. It should'nt be difficult to enable to change the remaining process according to a forzen past.

It implies providing more control on activations. Indeed, this is currently the more important class in term of responsabilities. It represents the realization sandbox and plenty of processing could be done with the conveyor. It is what represents the true realization of a planned activity (through a planned BP).

- it help manage interactions
- it manage recording of the execution (log++ replayable)
- it manage the agent ressource

2.6 Recording a workflow execution

This part is quite empty in the initial relase of Worflow. This was clearly coupled with the persistance solution (Omnibase).

Discussions will be summarized as a suggestion to make recording replayable.

The main classes dedicated to recording xxx and its sublasses. There is a stub method that should be coded (dependant on the persistance solution).

2.7 Scheduling a workflow

No scheduling at the moment. Clearly this is something to add, at least a naive version enabling activations management. One probably has to add to Activation class capabilities of suspeding, cancelling, ...

The workflow execution produces activations. These activation are associated to ressources.

We have to consider a ressource has a queue to process activation (say ithe agent is a person and has several opened work). The queue is the operating system between the agent that realize the activity (through activations).

To do use a scheduler, a planner ? suggestion. Provide execution of several workflows.

CHAPTER 3

WfWorkflow Core Design

To do

CHAPTER 4

Design Discussion

This chapter is about design decision needed to have a general purpose workflow engine in Pharo.

Un process is a succession of activitiesa ccording to conditions to accomplish an action, an objective. The definition of a process is therefore quite general. A process execution conforms to its process defintion (as much as possible). In real life, realization often derives from plans and adapt.

Workflow should enable worflow realization to eventually diverge when the realization occurs. Workflow realization in WfWorkflow is reified through activations.

4.1 Dynamic process execution

4.2 Persistance and import/export

Persistance is not only about backup. Especially in Aare, it was used

- backup, versioning, logging,

4.3 KISS - A generic solution

Voyage seems a perfect candidate to store obejcts. At start, all will be in image. We focus primarly on interactions.

The most visible remaining dependance of Omnibase is the class `WfManagedObject` that was initially a subclass of Omnibase `OBMangeObject`.

Voyage because you get an in memory storage and if needed a mongo or PUNQLite.

To do You could also save your objects using STON (serialization).

4.4 Interaction

One major challenge is to design a business process engine that is loosy coupled with interactions solutions like web user interface. The solution requires a loose coupling framework allowing to build simple interface with agents/users, being human or another system. Annoucment seems like an interesting framework to build on. Ideally, there should be:

- web interfaces (forms frame)
- Desktop interface (SPEC, Brick)
- services/messages communication bus

Useful abstractions to achieve such generic interoperability are:

- message-based interaction
- and/or event-based interaction
- and/or service-based interaction

To do A définir !

4.5 Design discussion

mesage-based interaction (communication network underlying)

Need to abstract communcation for the differents agents.

CHAPTER 5

Appendix

5.1 Business Process software comparison

WorkflowSystemsCompared

5.2 Screenshot of the initial web based interface