NewWave Workflow Engine

Sebastijan Kaplar
Faculty of Technical Sciences
University of Novi Sad
Novi Sad, Serbia
kaplar@uns.ac.rs

Miroslav Zarić
Faculty of Technical Sciences
University of Novi Sad
Novi Sad, Serbia
miroslavzaric@uns.ac.rs

Gordana Milosavljević
Faculty of Technical Sciences
University of Novi Sad
Novi Sad, Serbia
grist@uns.ac.rs

ABSTRACT

NewWave[1] is an open-source workflow engine written in Pharo. The driving idea behind the work presented in this paper is to create an extensible workflow management system, natively written in Pharo, to facilitate easy specification and implementation of application business logic. Workflows can then be executed in a controlled manner, with the ability to gather and process business data using highly customizable plugins. Although still in the development stage, NewWave engine already supports basic control-flow patterns such as sequence, parallel split and join with synchronization, exclusive split and join, several event types, etc. This makes the NewWave engine applicable for specific real-life workflow models.

KEYWORDS

Workflow, Workflow engine, Pharo, TaskIt

1 Introduction

Process-aware business systems have gained, in the last decade, strong support throughout the landscape of business information systems. Most of these systems are relying on a possibility to decompose business functions as a sequence of actions that, when executed in a controlled order, are driving the business toward a successful outcome. Most of the business functions may be represented as workflow - an orchestrated and ordered execution of activities. Reduced to its core elements, workflow technology serves to control and coordinate, in a flexible way, the execution of sets of well-defined operations [1]. Work scenarios are specified, executed, and controlled with the workflow system. This technology has found use in many different domains, including business processes, scientific applications, and e-learning. One of the main reasons for this expansion is that workflows employ concepts from the specific domains, which is why workflows are commonly used by domain experts without extensive technical background [1]. Although identification of discrete actions and their proper ordering is usually the most important and the most obvious part of the workflow model, for successful execution, workflow model (and execution engine) must also have the capability to express process data and task assignment model in order to be truly functional.

[1]https://github.com/skaplar/NewWave

Pharo, a purely object-oriented language, has much potential to implement and support many business-oriented applications. Pharo is providing simple, yet powerful syntax, and is focused on simplicity and immediate feedback, promising developers almost instantaneous results. Pharo is also dynamically typed, and many Pharo related projects are aimed at supporting model-driven and dynamic approach throughout all phases of software development: specification, design, and implementation. However, there is a lack of widely adopted native support for workflow modeling and implementation. This fact impairs the ability to have a process-driven application dynamics. After deprecation of AARE [2], a workflow management system written in Pharo initially released in 2005, a gap was opened in its place. NewWave was created with the idea to reintroduce native support for workflows in Pharo, and to facilitate easy process handling by leveraging the TaskIt[2] library.

Outline of this paper is the following. The motivation for creating a new workflow engine is described in Chapter 2. Chapter 3 describes the implementation details, while Chapter 4 contains conclusions and future work.

2 Motivation

Creating a workflow engine is a challenging task. There are many available options, on a variety of platforms, ranging from commercial solutions, partially commercial (where some advanced features are not available on free to use versions), as well as freeware solution (but usually with severely limited functionality). In Pharo language community, an idea to create an open-source workflow engine has emerged.

The Workflow Management Coalition (WfMC) has proposed a reference architecture, used as a blueprint for developing a fully functional workflow engine. While there is much work to implement all the standards and recommendations stated in the reference architecture, our goal is not to simply create yet another implementation of the workflow engine. The main idea is to design a flexible, extensible development environment which enables integration with other applications and tools, and also to be able to act independently, providing everything that a process-oriented application needs for its execution (dynamic generation of user interface forms for user tasks, database persistence of process definitions, active users, and active processes, etc.).

[2]TaskIt - https://github.com/sbragagnolo/taskit

Other ideas include distributing NewWave over multiple Pharo images and enabling the orchestration and communication between those images where every image would be given a certain type of task to perform, and lastly to achieve dynamic process reconfiguration without violating the workflow system. Although AARE workflow engine is not supported anymore, there are two more projects, built for different purposes:

- BPM Flow [3] which is built at the backend of GemStone/S from GemTalkSystems. BPMFlow is an open-source implementation of BPM standard and it includes: back office application, front office application, business intelligence applications, and a number of features. Unfortunately, there is no similar solution in Pharo.
- A4BP (Assessment for Business Processes) is a platform whose purpose is to craft custom analysis and obtain quality metrics of imported Business Process models already created with external tools. The main idea is to provide a tool to navigate the entire business process definition including the relation between process and technological services related to that process [4]. A4BP is based on the Pharo platform.

3 NewWave Workflow Engine

NewWave allows users to specify workflow composition. It enables specification of processes and activities as different types of tasks and ordering of those activities. Every workflow can be inspected with a simple yet informative visualization which is achieved using Roassal[3]. NewWave utilizes TaskIt, a library that enables the usage of tasks in Pharo, with abstractions to execute and synchronize concurrent tasks (Figure 1).

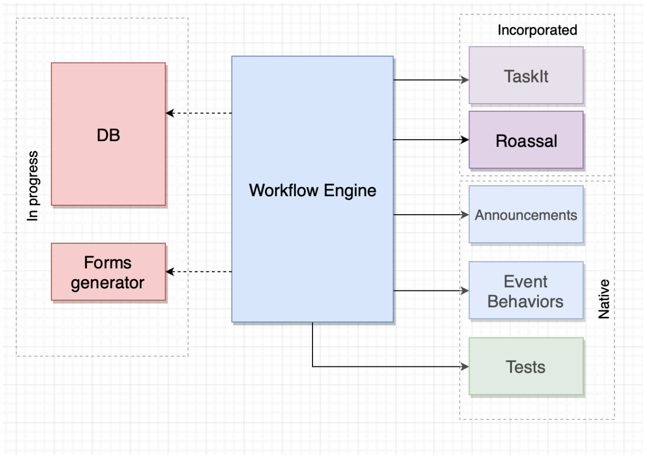


Figure 1. NewWave workflow engine dependencies

An activity is defined within the Business Process and represents work to be performed to achieve desired results. The activity usually takes some time to perform and requires one or more resources, some input (i.e., a form to collect data) and will produce some output. An activation represents one instance of the activity in the running workflow. The activity can have multiple instances (activations) in the same workflow.

WfMC defines a Reference Model [5] which proposes standards in order to foster interoperability between different workflow systems. NewWave supports an array of widely accepted business workflow patterns which will be listed. Some of the patterns not listed can be achieved by combining supported patterns.

Sequence or sequential routing is a basic control-flow pattern, where activities are executed in a sequence, and the current activity is enabled after completion of previous [5].

Conditional routing or the XOR-split represents a decision in the workflow process where the branch is forked into two or more branches, and the control is passed to one of the outgoing branches based on a specific condition [5].

Parallel routing or the AND-split represents forking of a branch into two or more branches where each of them is executed concurrently [5].

Synchronization or the AND-join represents meeting two or more branches into one subsequent single branch, and the activity from that single branch is run only when all previous branches have been completed [5].

3.1 Types of tasks

NewWave supports different types of tasks that can be performed. Elementary type of task is *BaseTask*. *BaseTask* has only one instance variable, which is *value,* and it is used to store and retrieve data.

ScriptTask derives from *BaseTask*, and it has one additional instance property, which is a *script.* The script property holds the block of code that should be executed when called. This is the reason why the *value* from *BaseTask* is redefined, and it looks like

```
ScriptTask>>value
        ^ script value
```

By sending the message *value* to the *script* variable, the block is executed and the foreseen operation is performed.

CustomTask allows for complex tasks definition. It allows embedding custom code and different libraries to gather data from the user. CustomTask also uses announcers to notify the engine when the task is completed.

UserTask allows to define a simple form for gathering data from the user. It is possible to use the API, provided by NewWave. to add different types of fields which will be displayed to the user during the workflow execution.

3.2 Designing the workflow

NewWave, as it is still in the early development phase, has no graphical workflow editor/designer. Nevertheless, specification of the workflow, for now, can be accomplished directly in the Pharo environment. NewWave has an API that allows defining the desired workflow (see Listing 1, for example). After defining the workflow, it is possible to use Roassal to get simple visualization

[3] Roassal2 visualization engine - http://agilevisualization.com

and inspect the designed workflow. An example of one such workflow is given in Figure 2.

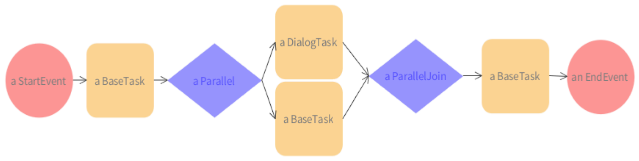


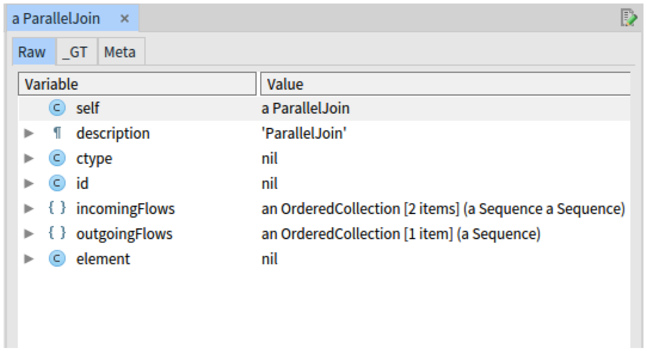
Figure 2. Workflow visualization with Roassal in NewWave

Figure 2 depicts a workflow where some common elements are shown. As we can see, we have a *StartEvent*, followed by a *BaseTask*. After *BaseTask* is executed, *ParallelSplit* is executed, and two tasks are performed concurrently. Those tasks meet at a *ParallelJoin* which is used for synchronization. Finally, one more task is executed before the workflow is completed. Creating the described workflow is presented in Listing 1. For simplicity purposes, the creation of some elements is omitted.

```
startEvent addOutgoingEdge: task1.
task1 addOutgoingEdge: split1.

split1 addOutgoingEdge: task2.
split1 addOutgoingEdge: task3.

task2 addOutgoingEdge: parallelJoin.
task3 addOutgoingEdge: parallelJoin.

parallelJoin addOutgoingEdge: task4.
task4 addOutgoingEdge: endEvent.

engine := WaveEngine new.
we := WaveExecutor initialNode: startEvent.
engine addExecutor: we.
```

Listing 1. Creating the workflow

It is also possible to inspect every element of the workflow using Roassal and Pharo builtin features. Figure 3. presents ParallelJoin, and it shows that it has two incoming flows, and one outgoing flow, which is also visible in Figure 2. Moreover, if it is needed, incoming and outgoing flows can also be inspected separately, for example, to show what elements are at the end of each flow.



Variable	Value
self	a ParallelJoin
description	'ParallelJoin'
ctype	nil
id	nil
incomingFlows	an OrderedCollection [2 items] (a Sequence a Sequence)
outgoingFlows	an OrderedCollection [1 item] (a Sequence)
element	nil

Figure 3. Inspecting ParallelJoin

Workflow execution

Workflow execution in NewWave starts by sending a message *startEngine* to the *WaveEngine* (Figure 4). This event message activates the *WaveExecutor*, which starts executing the previously defined workflow. Workflow execution can be logged and followed from start to finish. During the execution, the user can interact with different types of tasks, entering data in the forms and deciding the path of execution based on offered choices.

WaveEngine

WaveEngine is one of the main components and represents the engine of the workflow. Engine controls and organizes the execution of the workflow, with all runtime information necessary to achieve effective control of the execution, such as work list, activations, control of the multiple executors, etc.

WaveExecutor

WaveEngine has one main *WaveExecutor*, which is responsible for executing the workflow. When necessary, for example, when *ParallelSplit* occurs, multiple *WaveExecutors* can be spawned and they exist until the end of their execution, for example, until *ParallelJoin* ends.

WaveExecutor has the task to provide a proper environment to execute the current element. To obtain the element for execution, *WaveExecutor* is relying on services of *FlowHandler*. *FlowHandler*, as its name suggests, handles the flow of execution. It keeps the information about the element that is currently executing. After one element execution is completed, *FlowHandler* determines, from the Workflow definition, the next element to be executed. It also keeps the information which elements were executed and in what particular order. This "execution history" is handy for retracing the steps of execution, following up on the workflow progress. Simplified architecture of the NewWave is shown in Figure 4.

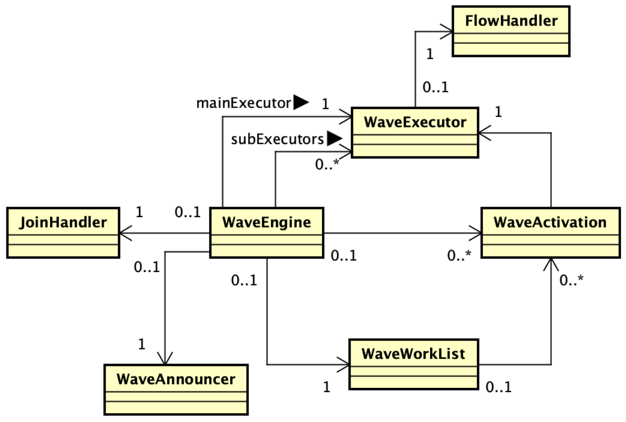


Figure 4. Simplified engine architecture

Tasks in NewWave, as it is a custom in most of the workflow engines, can be broadly classified as automatic or manual, and they can be combined through the workflow. Using the scheduling mechanism, tasks can be scheduled to perform at the desired time, with the option to repeat when necessary.

Activations in the NewWave

Activations API is a part of the NewWave engine. Every node (task or gate) in the engine has an activation associated with it. The activation becomes active when the engine tries to execute the current node. In that moment node is activated and information about it is recorded in the engine. When the execution finishes, the activation changes state to completed and updates the information in the engine, with all the additional data the engine needs.

This information can be used to query the activation for the nodes in the engine, to check in what state it is, or to get information about the executor instance running the node, and so on.

Using announcers in the NewWave

Pharo Announcement framework is a small but powerful event notification framework which provides the generic implementation of Observer pattern. Using announcements simplifies event system in NewWave and provides customization options for executing different types of events. For example, we can consider synchronization, where multiple branches converge into one. In order to continue workflow execution past join point, all previous parallel branches should have been completed. Every time a branch reaches the synchronization point, a *JoinEvent* gets announced, and it is used to check if the synchronization is completed. Announcement framework allows for customization so that the Announcement class can be subclassed (see Listing 2).

```
Announcement subclass: #JoinEventAnnouncer
        instanceVariableNames: 'parameter executor'
        classVariableNames: ''
        package: 'NewWave-Announcers'
```

Listing 2. Describing the JoinEventAnnouncer

JoinEventAnnouncer has two instance variables, a *parameter*, and an *executor*. The *parameter* represents a sequence that leads up to synchronization point, and the *executor* is the *WaveExecutor* instance running this execution. Since the Announcement allows events as objects, this particular behavior is used to notify the part of the engine which handles synchronization with the objects that are being executed. NewWave offers support for different kinds of events and allows embedding new types of events; the more common ones are shown in Figure 5.

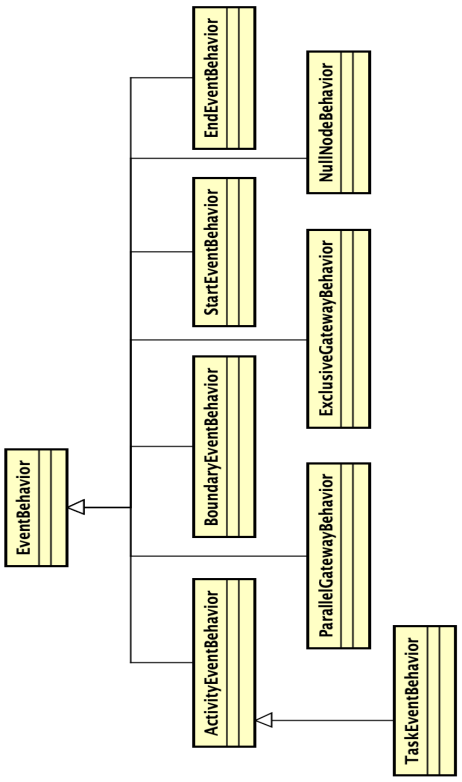


Figure 5. Events supported in NewWave

NewWave uses TaskIt, a library written in Pharo, that abstracts execution and synchronization of concurrent tasks. NewWave utilizes two mechanisms, in particular, the worker runner and futures. The worker runner is a task runner (instance of *TKTWorker*) that uses a single process to execute tasks from a queue, and with workers, NewWave can control the number of live processes as well as how tasks are distributed amongst them. Future is an object that represents a future value of the task's execution. When the task completes, the result is deployed into the corresponding future, thus allowing access to its value.

However, it can not be accessed right away, and futures provide callbacks to access its value asynchronously.

4 Conclusion

NewWave project was started with the idea to reintroduce native support for workflows in Pharo. This paper presents the first steps towards the goal to create an extensible workflow management system to facilitate easy specification and execution of application business logic. Currently, NewWave supports:

- Control-flow patterns (sequential routing, parallel routing, conditional routing, synchronization)
- Different types of tasks (user tasks, custom tasks, script tasks)
- Scheduling
- Different events (start event, end event, boundary event, custom engine events)
- Subprocesses
- Specification of workflows using the NewWave API
- Visualization and inspection of the specified workflows
- Pharo Announcements framework.
- Rudimentary support for user interface forms for obtaining data for user tasks.

Future work involves: (1) development of graphical workflow designer, (2) better support for user interface forms for obtaining data from user tasks, (3) serialization of process definitions, active users, and active processes in a database, (4) importing BPMN (Business Process Model and Notation) diagrams, (6) support for the integration with various tools through plugin mechanism, and (7) distribution over multiple Pharo images and enabling the orchestration and communications between those images.

ACKNOWLEDGMENTS

The authors would like to thank Stéphane Ducasse and Cédrick Béler for their kind help and support throughout the project.

REFERENCES

[1] Puccini, Mario Eduardo Sánchez. "Executable models for extensible workflow engines." PhD diss., Uniandes, 2011.
[2] "AARE - Workflow Management System", Whitepaper, Netstyle.ch GmbH, 2005, https://github.com/Netstyle/Workflow/blob/master/Workflow-whitepaper.pdf
[3] BPM Flow Manual, https://bpmflow.gitbook.io
[4] Peralta, Alvaro Jose, Nguyen Tuan Thanh Le, Serge Stinckwich, Chihab Hanachi, Alexandre Bergel, and Tuong Vinh Ho. "A Tool for Assessing Quality of Rescue Plans by Combining Visualizations of Different Business Process Perspectives." In International Conference on Information Systems for Crisis Response and Management in Mediterranean Countries, pp. 155-166. Springer, Cham, 2015.
[5] Workflow Reference Model, http://www.wfmc.org/2-uncategorised/53-reference-model