# A Generic Framework for Workflow Definition and Execution

Anindo Ashim Saha
Department of Computer Science
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, NY 14586
as3601@rit.edu

***Abstract*—The design of software is a subjective matter, which depends on a lot of functional and non-functional factors and these factors have an effect on reusability, maintainability and development velocity. However we can still view the design of any software as a bunch of heterogeneous components and each component requiring certain inputs, performing certain actions, and emitting certain outputs. These components can be arranged in a graph of dependencies in a way to accomplish a certain activity or a business process.**

**A workflow engine is a software that is used for defining business processes and executing them. A major goal of every workflow engine is to be able to define workflows in the easiest way possible without sacrificing flexibility or power.**

**We include in this report, a survey of existing workflow engines, their features and their approaches. We also show a way to define individual components, each conforming to a specific interface and arranging them into a graph in terms of their dependencies and automating their execution. As a result we come up our own workflow engine which can define workflows as directed acyclic graphs and a way to execute them.**

***Index Terms*—workflow; business process**

## I. Introduction

Workflows are used to design the steps that compose a business process. Each step in the workflow encapsulates a small part of the overall process. By modularizing in such a way, we can keep a tab on the increasing complexity of software and also increase reusability. Every step in the workflow if devised carefully can not only be developed independently but also be reused whenever required as part of another business process. Our software aims to to force modularity by providing a framework to define the individual steps and the overall workflow and to enable execution of such workflows. We aim to design our framework to make it trivial to develop simple applications, and provide a way forward to develop complex applications only in terms of the business processes they serve.

## II. Related Work

A number of workflow engines exist with varying levels of feature completeness. There are workflow engines which only focus on execution of a data processing pipeline and scheduler capabilities. On the other hand, there are workflow engines, better known as BPM(Business Process Management) suites which handle end-to-end automation of the business process and provide persistence and messaging capabilities. In this section we will look at 3 different software, jBPM, Cromwell and Airflow each having different approaches.

### A. jBPM

jBPM [1] is used for building business applications for automating business processes. jBPM is open source software, written in Java, released under the Apache Software License. It is used for defining, executing and monitoring of business processes.

jBPM [2] is used as standalone service or embedded in other Java applications. It can be used in JEE applications as web appplication archive(war) deployments included as a library in standalone java programs. Other than its core purpose, it also has other components which include data modeller, reporting tools and a rules engine.

jBPM is based on the Process Virtual Machine library by JBoss and uses the BPMN 2.0 notation to store and define business process definitions. Earlier versions(before jBPM 6) used to use jPDL(Java Process Definition Language) for describing workflows. It uses components like Activity, Events and Gateways to structure the process definition.

Every execution of a process is called a process instance. Some activities are automatically executed while some require human/external intervention. jBPM keeps a track of the process state at every step by persisting the state at every change.

It is mainly used to replicate business processes in organizations and provides a good bridge between developers and non-technical users with its support for GUI tools for defining workflows.

Some good ideas from jBPM are its focus on non-technical users, design based on the Process Virtual Machine concept and abundance of auxillary tools provided like data modelling tools.

### B. Cromwell

Cromwell has been design to execute scientific workflows. It focuses on simplicity in process definition by using a human-readable domain specific language called Workflow Definition Language.

Cromwell can be used in two modes. In run mode, Cromwell executes a single workflow and exits upon com-

pletion of the workflow returning an exit code. Run mode is good experimentation and development phase.

Cromwell supports server mode for more advanced use cases involving large workflows. In server mode, Cromwell starts as a web server that exposes REST endpoints.

Cromwell supports the concept of configurable backends. A backend in Cromwell means an environment to run the workflow. Cromwell has backends ranging from local machine, HPC clusters and public clouds.

Some good ideas from Cromwell are domain specific language standard that is independent of the workflow engine[2] concept of run and server modes and its focus on running scientific workflows.

### C. Airflow

Airflow is a platform to programmatically author, schedule and monitor workflows.

Airflow is used to author workflows as directed acyclic graphs (DAGs) of tasks. The scheduler executes tasks on workers based upon dependencies. The user interface makes it easy to visualize pipelines running in production, monitor progress, and troubleshoot issues.

Airflow pipelines are Python code allowing for dynamic pipeline generation. By defining our own operators, executors we can cover many usecases. It has a modular architecture and uses a message queue to scale up job

Some good ideas from Airflow are its use of a programming language for defining workflows, concept of operators and the extensibility it provides and the multiple extensions with respect to the execution environment and other concerns.

## III. Design and Implementation

The primary unit of organization to define a workflow in our workflow engine is a *Task*. Every *Workflow* is just a directed acyclic graph of dependencies among *Task*s. All tasks must implement the Task interface and provide a unique identifier. *Task*s also need to provide a name parameter, which is human friendly way to identify a task. A *Workflow* can be constructed using the *WorkflowBuilder* interface, which incrementally constructs the dependency graph of *Task*s. The *WorkflowBuilder* has methods like *addPipe* and *addFanIn* to create dependencies among *Tasks*.

Once a *Workflow* is completely constructed, it can be used by instantiating it. We define this term as a *WorkflowInstance*. A *WorkflowInstance* clones the graph of *Task*s. Every *WorkflowInstance* contains a map of variables that are part of its state. A *WorkflowInstance* can then be started using the *startWorkflowInstance* method which computes the first *Task*s of the *Workflow*. The *executeWorkflowInstance* method can then be used to execute a particular task. On every execution of a *Task*, it is supplied with *WorkflowInstance* and *Task* specific variables, which are used by the *Task*. The *Task* can also inject new variables in the *WorkflowInstance*. The *WorkflowInstance* is completed when all the *Task*s have been executed.

## IV. Upcoming Work

In the upcoming weeks as part of milestones 2 and 3, I plan to refine the existing design and implementation and introduce a file-based format to persist state between workflow instance executions, implement nested workflows and allow multiple instances of such embedded workflows to exist in a parent workflow execution instance.

## References

[1] R. Doe. (2009, Jun.) jbpm documentation. [Online]. Available: https://docs.jboss.org/jbpm/release/7.17.0.Final/jbpm-docs/html$_single/$
(2009, Jun.) This is a entry of type @ONLINE. [Online]. Available: http://www.test.org/doe2/