# Workflow Engine Evaluation

**Product**: \_\_Ptolemy / Passerelle / Triquetrum\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**URL:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Evaluator:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Date:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Instructions: Fill out the above information. For each item below, give a score from 1 (does not meet requirement) to 10 (fully meets requirement) in the blank right after the bullet. Then provide supporting info in the space following the item’s text, if possible.*

1. **\_\_10\_ Open source.** The product must be free to use, and should offer source code access. A viral GPL license would be a problem; prefer an Apache or BSD license that does not limit our options. Please state license in the space below.

Ptolemy has a BSD style copyright, Triquetrum will be EPL.

Passerelle is Apache 2.

1. **\_\_10\_ Supported product**. Should be an active, thriving project. Should have corporate sponsorship as evidence of lasting support.

Ptolemy has a long and active history at UC Berkeley.

Passerelle is supported by iSencia since 2002, with collaborations with some european synchrotrons.

1. **\_\_10\_ Portable**. Must run on Windows, Mac, Linux, and NGP (which is just modified Linux, one imagines.)

Requires a JDK7 (currently) and equinox, SWT etc, which should all be fine.

1. **\_\_9\_ Open standards.** Usage of documented open standards for workflow storage, communication protocols, query language, graphics, etc is a plus. Proprietary/closed standards are a big negative, especially binary file formats.

Workflows are stored in XML files (ptolemy's MOML DTD).

The task-based processing domain model has a JPA-annotated implementation, typically mapped to a standard RDBMS.

There are services and actors to communicate via different protocols like standard sockets, FTP, SOAP, HTTP, JMS, ...

Gets a 9 because MoML is not used in other products

1. **\_\_10\_ Java-based.** Engine itself must be Java-based, or Java friendly, as we’ll want to embed it in Java, and extend it with Java.
2. **\_\_8?\_ Python-friendly**. At a minimum it should be easy to invoke workflow actions consisting of Python scripts.

Thanks to work done in DAWN, there is an actor and service layer to invoke Python scripts. They have also done a lot of other work to integrate Python in different ways in a scientific workbench.

Gets an 8 because I’m not familiar with how DAWN and Python work together. Feel free to update.

1. **\_\_5\_ Graphical builder.** The product should have a full-featured graphical workflow builder. A big plus if builder is SWT-compatible so it can be integrated in SAW.

 This is one of the primary goals of the Triquetrum project.

 Passerelle already has a basic GEF-based editor.
 The intention is to make a better one based on eclipse Graphiti (or a similar fwk).

1. \_\_\_?\_\_ **Data tools.** The graphical builder should include tools for extracting data from files as part of workflow, like iSight

There are many options to load/extract data from different file formats and use the data in workflows, also in streaming mode, lazy-loading etc, i.e. where the total data size does not fit in memory.

But I'm not sure what is meant with “tools in the graphical builder”.

Could you maybe illustrate with screenshots or so (e.g. from iSight)?

1. **\_\_8\_ Easy to implement workflow actions**. The engine must be able to execute external programs as actions. It should be able to use “plugins” written in Java. “Plugins” written in Python would be nice to have. It must have well-documented API or mechanism for defining actions.

Ptolemy has a fully documented Actor dvp API. Python (and JavaScript) may be used to develop Ptolemy functional blocks.

Passerelle has added APIs to make actor dvp even easier,
and/or implement Task-based actors. New actors can be dynamically added to an existing workflow runtime without downtime.

Triquetrum will also offer this.

Gets an 8 because the current UI is a bit clunky and the new UI has not been developed yet.

1. **\_\_1\_\_ Abstract data channels.** Must support the notion of an abstract data channel, explicitly or implicitly. The number one consequence of this is that exchanged data mustn’t need to pass through engine or through a file. The builder should support explicit “type” for data links between workflow actions.

This requirement is not completely clear for me (cfr also req 8)

Passerelle has been integrated with several data channel systems. There is no a-priori requirement that the data must be sent between actors inside the workflows or engine.

Ptolemy has an advanced type-validation system to check that actors can exchange matching data types. Ptolemy does not directly support abstract data channels.

1. **\_\_5?\_\_ Workflow persistence**. Workflows should be persistent during execution to facilitate restart, progress feedback, and reporting. Bonus points if there are existing reporting tools that support getting workflow status on your iPhone.

Current open-source Passerelle defines the entities for storing detailed execution traces, task lifecycle events etc in an RDBMS.

But the fully integrated system, incl web views, querying etc is not part of that.

This could be part of extra features in Triquetrum, potentially reusing some of the work in DAWN, as shown by Matt during the call.

A web-UI would be ready for your iPhone. We do have a REST service API to start/stop/suspend/resume flows and get status, but not yet an app for iPhone on that. Could be an interesting addition.

Kepler (http://kepler-project.org), which uses Ptolemy as its engine has some of these features. Kepler is also BSD.

1. **\_\_10\_ Workflow storage**. Builder must save workflow definition in a portable file, and engine must read entire workflow definition from this file. Files should have an open/standard format. We should be able to save workflow action definitions in a “library” for reuse.

Stored in XML files (cfr req 4).

Ptolemy has advanced features for sharing models via user model libraries. Ptolemy supports actor-oriented design, which is similar to object oriented design where model designers develop actor classes that are instantiated and extended. Passerelle adds APIs for model repositories that would be part of Triquetrum as well.

1. **\_\_3\_\_ Scalability. Engine and builder** must handle hundreds or thousands of nodes in a workflow. Engine must support running multiple workflows simultaneously. Engine must work with exascale problems.

We have models with hundreds of actor nodes. Not yet thousands.

We should check with Ptolemy about this, as we do see that a complex instantiated model can take several Mb of memory. Ptolemy has been invoked on models with 20,000 actors. We have generated C and Java code from these large models. Ptolemy supports lazy instantiation of composites, which makes browsing large models easier. Scalability is probably the biggest area for possible improvement in this list.

We have installations that run > 100K workflows per day in a cluster of 10 nodes. Server runtimes can run many flows concurrently.

An eclipse RCP workbench is typically not made for that, but could be a client to edit/upload/execute/monitor workflows on a server.

1. **\_\_10\_ Headless mode.** Workflow execution must support headless (no-GUI) mode on all platforms

Workflow runtimes can be started headless from the cmd-line to run one specific flow, or can be running as servers. For runtime servers, a REST API can be used to upload/start/stop/suspend/resume workflow executions.

1. **\_\_10\_ Composable workflows.** Engine and builder should have support for nested or composable workflows

This is again a core feature of Ptolemy. Composite components can be defined, instantiated and extended.

1. **\_\_4\_ Distributed operation.** Engine should support distributed operation. Just executing actions via ssh would be fine, as long as we can leverage Kerberos authentication.

This is done by using actors that invoke external actions.

Many options are available : SOAP, JMS, HTTP+XML, JMX, …

There is also an integration with DRMAA to run operations on computing grids (e.g. on SGE, SLURM, …).

More options are available, but not yet in open source. E.g. automated clustering of engine instances on Hazelcast to do load distribution, distributed memory etc. This can be discussed in more detail.

Partitioning models is an area of open research. If a model is distributed incorrectly, then it will run more slowly than on one machine because of communication issues.