Generic Workflow Engine Design

Peter Pavek
2011-12-21

A workflow is a sequence of actions. It can be thought of as as a lattice consisting of nodes and arcs. The arcs are directed and imply sequence between the nodes. The nodes carry action information, i.e., what to do, and parameters. The actions -- at an abstract level -- include I/O, computation, and routing directives. The workflow engine picks a node, executes its actions and finds the next node to execute. When it runs out of nodes to execute, it stops.

The workflow engine also passes data between the nodes' actions, so that the results of one node can be used in another node.

A workflow is implemented as nodes -- instances of a class -- and connections -- pointers to the next nodes. The node classes have a number of standard methods;

- `evaluate` -- runs the block's action.
- `getNextBlock` -- given the results from evaluate, returns the next block.

The data passing is implemented as a sequence of variable-name / value pairs. This sequence, called "context", is passed to the blocks' methods. They can read or change it.

The workflow engine algorithm is as follows;

1. Get first block BL.
2. result = evaluate(BL, context)
3. if result == error then stop
4. BL = getNextBlock(BL, context, result)
5. Go to step 2

This is implemented as reentrant code, where each invocation (process instance) has its own context data.

The algorithm above can be extended by adding tracing, logging, and bookkeeping. For example the following block methods can be added and called in the core algorithm loop:

- `logNodeEvaluation` -- logs the block's actions.
- `animateWorkflowProgress` -- shows progress of workflow to the current node.

The algorithm is then extended to the following;

1. Get first block BL.
2. animateWorkflowProgress(BL)

3. result = evaluate(BL, context)
4. if result == error then go to ...
5. BL = getNextBlock(BL, context, result)
6. Go to step 2