Content Routing

Specification — 2012-05-02

Authors:

- Alain Escaffre
- Anahide Tchertchian
- Mariana Cedica
- Florent Guillaume

Lexicon

Workflow instantiation:

- **Workflow definition**: the abstract definition for a workflow, expressible in XML, created in Studio and imported in Nuxeo.
- **Workflow model**: a concrete model for a workflow, materialized as a set of documents in Nuxeo.
- **Workflow instance**: a concrete model for a started process instance, materialized as a set of documents in Nuxeo.

User roles:

- **Workflow designer**: designs and configures new workflows in Studio.
- **Workflow manager**: is allowed to manage workflows in Nuxeo in the Admin Center.
- **Workflow user**: any end-user who is assigned some rights during a workflow instance execution.

Workflow execution model

To understand the various pieces of information needed when creating a workflow in Studio and when managing it and using it in Nuxeo, it is important to describe the workflow execution model and what it required in terms of data model.

Graph

An instance of a workflow is a set of nodes. Each node has a list of nodes that it can transition to. Each transition is associated with a condition. The condition is evaluated according to the context variables (see below).

A workflow has a starting node, and one or more stop nodes.

In the graph we distinguish two kinds of transitions: normal ones and *loop transitions*. When following a path of transitions from the start node to a stop node, if at some point a transition moves back to a node already part of the path then it is said to be *looping*. A transition is a *loop transition* if there exists at least one path where it is *looping*.

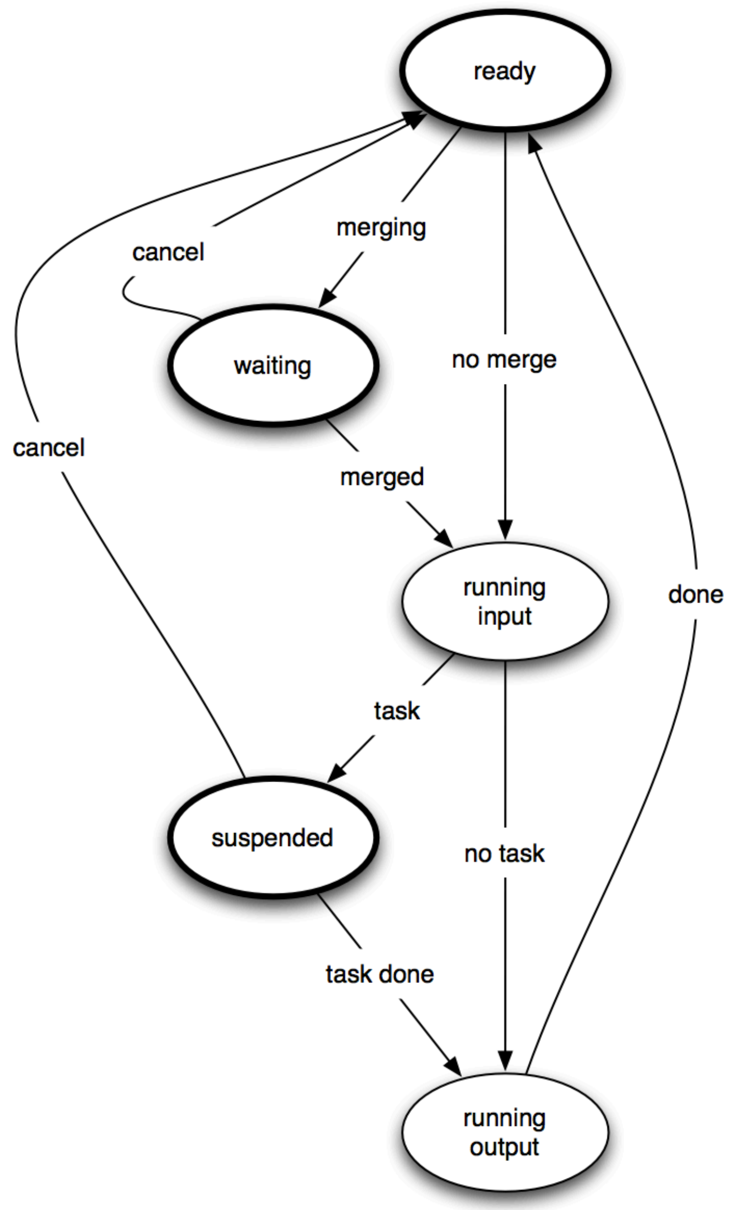
The nodes are materialized as documents, stored in a container that materializes the workflow instance.

The workflow instance may have a set of variables used for the node context but which are shared by all node contexts and stored at the workflow instance level.

Node state and context

During execution, each node has a state (lifecycle state):

- *ready* (default state for all nodes)
- *waiting* (for merge nodes)
- *running input & output* (internal state not persisted)
- *suspended* (when a task has been created)



When a node is *done*, a counter on the node is incremented,

When a node is *canceled*, a *canceled* flag is set on the node.

Each node has a set of runtime context variables, stored in the node context. There may be variables defined at the workflow level for all nodes, and variables defined just for one node. The context variables have a value that can be defined or changed:

- as an initial value set at workflow definition time,
- as an operation context variable for the node chains,
- when a form associated with a task is validated.

Some special variables are needed to represent:

- the document(s) associated to the workflow,
- the input and output values of an operation chain,
- the chosen exit status for a task,
- the initiator of the workflow,
- a task id? (or reference from the task to the node?)

The successive values of the node context variables are saved in the document audit for later retrieval.

When a node has several possible input transitions, it is a merge node. Such a node must define a merge condition based on when its input transitions are followed:

- wait for all
- wait for first
- …other (n out of m)?

When execution first arrives at a merge node, its state is set to *waiting*, and will change to *running input* only if the merge condition becomes true. If the merge condition becomes true before all input transitions are followed, then the nodes above all non-followed input transitions have to be canceled (including their associated tasks).

When a node decides to create a task, the node state is set to *suspended*. When the user clicks on the task form to complete it, form information is saved in the context and execution resumes at the output chain for the node.

Adding nodes dynamically

Some use cases require that a workflow user be able to add one or more nodes dynamically after the current task's node, to make the workflow go through additional people. These added nodes are serially added in the middle of one of the transitions from the current node to the next.

The new nodes may have additional reject-like transitions added as well, we can see 3 choices for these reject transitions:

- reject as same transition as in the original task node,
- reject to previous node,
- reject to next node (just marks the node status as rejected but continue the workflow).

The kinds of nodes that can be added dynamically will have to be configured on the base node. They may be nodes that invoke fixed chains (email notifications for instance), or nodes that create new tasks (validation).

Runtime node information

A node document representing the runtime state of a node needs the following data:

- id (optional short string, generated at design time if missing)
- label (for humans)
- description (for humans)
- state (lifecycle state)
- node context variables and values
- input chain (optional)
- output chain (optional)
- list of exit transition with for each:
  - id (short string)
  - target doc id (not just node id, this is pre-resolved, so use the id of the doc representing the node in the workflow instance)
  - label (for humans)
  - description (for humans)
  - condition expression, either:
    - hardcoded ${status} == ${transition}
    - an arbitrary expression
  - condition evaluation result (true/false)
  - transition chain (optional)
- task creation (optional):

    - task creation flag
    - directive (for humans)
    - description (for humans)
    - due date (optional)
    - assignees (to decide to whom the task is assigned), either:
        - fixed list of users and groups,
        - or (implicitly) the value of ${assignees}
    - assignees ACL definition, either Read or Write
    - task form layout,
    - task completion buttons (defining the value of ${status} on execution), which is a list of:
        - id
        - label
        - filter to decide when it is displayed
- merge style (one/all)
- stop flag
- dynamic workflow modification
    - a flag
    - a filter definition to decide who is allowed to add new task
    - the list of allowed node model to be added, with, for each, the behavior after the node is processed (see the upper list of choices)
- start date (for audit)
- end date (for audit)
- canceled flag (for audit)
- counter (for audit)

Runtime context variables

The following context variables are available during workflow execution. They may have an initial value set at workflow definition time on the workflow or the node, or set by context merges.

- **workflowId**: the workflow id (short name),
- **initiator**: the workflow user initiator,
- **documents**: the document(s) associated to the workflow instance,
- **workflowStartTime**: the workflow start time

For a node, the following is defined:

- **nodeId**: the node id (short name).
- **state**: the node state,
- **nodeStartTime**: the node start time

For a task node, the following is also defined:

- **assignees**: the list of assignees to which the task is sent (computed by input chain),
- **comment**: a predefined variable to receive a form comment,
- **status**: the task exit status, defined according to the chosen task completion button.

During evaluation of a transition condition, the following is also defined:

- **transition**: the transition id.

In addition, the global workflow context and the node context variables are also available during execution.

Workflow engine algorithm

This section defines the logic implemented by the workflow engine.

All state is held by the node instance documents and the workflow instance document.

The workflow engine loop uses as internal state a set of *pending nodes* to process. When the workflow engine is called, the initial set of *pending nodes* is a single node:

- either the start node when starting a new workflow,
- or the resumed task's node when resuming a workflow from a completed task.

The workflow engine loop runs until there are no pending nodes in the list; for each one it decides what to do depending on its state:

- ***ready***:
  - if this is a merge node, set state to ***waiting*** and continue there,
  - otherwise set state to ***running input*** and continue there.
- ***waiting***: check all the input transitions results to see how many are *true*, then decide depending on the merge style if this merge happens:
  - if there is no merge yet, leave state to ***waiting*** and continue the workflow loop,
  - otherwise do the merge:
    - recurse on all nodes from incoming non-loop transitions to cancel them:
      - set the node's *canceled* flag,
      - if the node was ***suspended***, cancel its related task,
      - move the node back to state ***ready***,
    - set merge node state to ***running input*** and continue there.
- ***running input***:
  - execute the input chain,
  - if there is a task, create it and set the node state to ***suspended*** and continue the workflow loop,
  - otherwise just continue to ***running output***,
- ***suspended***: set the variables from the task form (this node is the one through which the workflow was resumed), and continue to ***running output***,
- ***running output***:
  - execute the output chain,
  - evaluate all the output transition conditions and store their result
    - for each condition that is true, add the target node to the pending nodes.
  - increment the node counter,
  - set the node state to ***ready,***
  - if the node has a stop flag then stop the workflow.

When a merge node cancels nodes from incoming non-loop transitions, the goal is to cancel the tasks of all suspended nodes that would eventually have transitioned "normally" to this merge node. To do this the graph is traversed backwards, but excluding loop transitions, and all tasks found are cancelled.

If the target node of a *true* transition is in the *suspended* state, then it is a workflow execution error (bad workflow definition). In this case the target node is not added again to the pending list.

If the workflow is stopped by a stop flag and there are still pending nodes, then it's a workflow execution error (bad workflow definition, there should have been a merge to cancel other tasks).

Creating a workflow definition in Studio

A workflow designer can create in Nuxeo Studio a new workflow definition. He needs to give:

- the id (unique) and label (human-readable) of the workflow definition,
- some filtering rules to decide on which document types this workflow can be started,
- the variables (name and nature) that will be global to the process execution,
- the graph definition of the workflow, using a graph editor.

The workflow designer must also have the possibility to copy and delete workflow definitions.

Workflow filtering rules

The workflow designer can specify a filter that will be used to filter out the workflows available to a workflow user when he wants to start a workflow on a document.

This filter is a list of allowed document types on which the workflow may be started. It may be extended later to be a more complex filter condition.

(A workflow model instantiated in Nuxeo can further be assigned a specific security through ACLs to allow it to be seen only by specific users.)

Workflow variables

Some workflow variables can be defined globally to the workflow.

The variables are defined in the same way that a schema is defined in Studio (regarding user interface), defining at least name, type and multi-valueness. Variables provide a set of values that will be known and available through the context by all workflow nodes, and can be updated by the nodes themselves according to operation chain computations or user input in tasks.

They aren't tied to a specific layout at this stage, this just defines which ones are available.

Workflow graph

The graph editor should provide a drag and drop interface to create new nodes, and provide a way to easily move and connect nodes together. The editor should also provide a list of predefined nodes (templates) to be used to implement the process business logic. Some templates (depending on their use) may require the entry of specific information before the node is created.

The workflow designer can choose among a list of built-in node types in his node library that can be dropped into the graph. The nodes are then connected together using simple arrows. All nodes are based on the same underlying model (see related section), but the use of simpler node templates allow the UI to specialize them for easier understanding by the workflow designer.

Node information

The following is the information that can be defined on a node. Templates can restrict which fields are predefined and which are editable for some node types.

- id (optional short string, generated if missing)
- label (for humans)
- description (for humans)
- node context variables and values
- input chain (optional)
- output chain (optional)
- task creation (optional):
    - directive (for humans)
    - description (for humans)
    - due date (optional), either:
        - from context variable ${due_date},
        - or relative from workflow or task start time (UI needed for choice of the time interval).
    - assignees variable name (to decide to whom the task is assigned)
    - assignees ACL definition, either Read or Write
    - notification template id (for assignees) (variable ${notificationTemplateId}?)
    - task form layout,

    - task completion buttons (defining the value of ${status} on execution)
        - id
        - label
        - filter to decide when it is displayed
- merge style (one/all)
- stop flag
- dynamic workflow modification flag
    - and the allowed modifications... TODO

The following is also defined on a node, but edited by a UI action on the connector:

- list of exit transition with for each:
    - id (short string)
    - target node id
    - label (for humans)
    - description (for humans)
    - condition expression, either:
        - hardcoded ${status} == ${transition}
        - an arbitrary expression
    - transition chain (optional)

General node information

The id, label and description are freely editable. Ids must be unique for a given workflow.

Node context variables

A node can defined new context variables in addition to those defined by the workflow. These variables are used to store information on the node during execution when this information is relevant only for one node. They can also have a default value.

Node Automation chains

The various chains that can be defined can be used in many ways, for instance:

- Input chains can be used to compute the task assignees in a specific manner, or to set some rights on a document.
- Output chains can be used for processing user input if a form was used, modifying rights once again, launching another operation chain, etc.
- Transition chains can be used for specific notifications only if a given transition was followed.

Node task assignment

Tasks are assigned to the users and groups corresponding to the value of the context variable *assignees*.

This variable can be filled:

- statically by defining a list of users and groups (behind the scene this actually defines a node context variable named *assignees*),
- dynamically be letting the input chain compute the value of the variable.

Node task form

A task node might have a form that is designed by workflow designer but filled by the workflow user during workflow execution.

The task form represents the UI that is available to the workflow user when he processes the task, it starts with a section displaying the directive, description and due date, and it allows the user to input data when a task is executed.

The task form is defined by a layout. The data entered is stored on the workflow as context variables. The form buttons are the task completion buttons:

- id
- label
- filter

Node escalation rules

User declares optionally a list of rules with time based triggering of automation chains. This can be used to configure sending of simple reminders. TODO

User declares a list of escalation closing status. This, when used with the above-mentioned rules, can be used to configure more advanced escalation scenarios where the process takes a different way than the normal one. (more details should be given, but not the good place here).

Dynamic workflow modification control

During execution of a workflow, when a user is assigned a task and before he processes it, he may have the right to add new nodes serially after his current task node.

The workflow designer defines a list of nodes that are displayed to the user when he is editing the live instance.

(It would be good to be able to specify a filter here to determine who can modify the workflow instance at this stage. In a first implementation, we can consider that it’s only the assignees and the workflow initiator who can add nodes.)

The new nodes are added after one of the output transitions of the current task node. These nodes’ output transitions have to be defined, we can assume that initially they will all be accept/reject transitions and form buttons.

Reject can come in 3 forms:

- reject to the same target as the current task node,
- reject to parent node,
- reject just sets the status but moves to next node.

Node merge style

When a node has more than one incoming transitions, the execution can either choose to proceed as soon as one transition is followed, or may choose to wait for all of them. This is controlled by the merge style flag, which can be *one* or *all*.

Node stop flag

When execution arrives at a node with the stop flag set, execution stops (the input chain is still run). There must be no output transitions when this flag is set.

Nodes library

The node library has a list of predefined node templates. When one node template from the node library is dropped in the graph, it creates a new node that can then be edited, but not all possible node fields are editable (or even shown), and some can be pre-filled. Also each node template may have a different graphical representation in the graph.

Once a node template is dropped in the graph and edited, there should still be a way to make the node “generic” by unlocking all hidden or read-only fields for advanced graph designers.

The node library should at least contain:

- a simple node with a task,
- a simple node without a task.
- an acknowledge task (with one closing status “Acknowledged”),
- an accept/reject task (with two closing statuses),
- a simple notification (predefined input chain),
- a notify and change lifecycle (predefined input chain),
- all CMF frequently required nodes (distribution, …),

There will also be some nodes used for structure:

- a 2-fork node (two defined output transitions with all conditions true),
- a 3-fork node,
- a switch node, with two transitions taken depending a variable being true or false,
- a decision node, with the transitions followed depending on a variable’s value.
- a one-merge node,
- an all-merge node,
- a stop node.

Creating new node types

An advanced studio user can extend the node library by creating a new node template. Creating a new node template is exactly like editing a generic node, but in addition each field has an additional flag:

- hidden,
- read-only,
- or editable.

This flag is used to decide what parts and how the UI displays this node when created in the graph.

The “make generic” action works as if making everything editable.

For simple node information making it hidden, read-only or editable is easy. For lists it must apply to the whole list, and in addition the controls to add or delete new elements on the list must also have a hidden/active flag. This allows for instance the display of two editable output transitions without the possibility of removing or adding new ones.

Automation requirements

Some additional operations will be defined for use by the workflow to be included in input, output or transition chains:

- Fetch Documents bound to current workflow. (Document, Documents): for use for instance by

notification operations.
- Start Workflow: enables to start a workflow given a workflow definition.
- Close Task (task name, closing status): can be used to automatically close a task.

Working with workflows in Nuxeo

Workflows in the Admin Center

In the Admin Center, the workflow administrator sees:

- a list of workflow definitions, defined by the Studio JARs that contain suitable extension points,
- a list of workflow models available to users as “route” documents templates.

Each workflow definition can be visualized as a graph in read-only mode.

The workflow administrator can ask the creation of a new workflow model from a workflow definition.

For each workflow model, the workflow administrator can also see the list of running workflow instances and visualize their history.

Starting a workflow

A workflow on a document may be started in two ways:

- Manually by the DM user: on the summary tab of a document, a “start workflow” widget is available and lists all the available workflow definitions allowed for this document instance. A start button allows starting the workflow.
- Automatically: An API allows starting a workflow. It should be possible to do it with Java code using Content Routing service, as well as with an dedicated operation that accepts a document and has as a parameter the name of the workflow definition to start. This operation can be use in studio to be bound to a custom button, or an event handler...
  Using the API, a workflow can be started with no document in the context.

Starting the workflow does not necessarily display a form to fill some workflow variables, but if one is needed the workflow should start with a task node that asks for it.

Once a workflow is started, the document for which the process is started has a new “Workflow” tab that displays the graph state and other relevant information.

Dashboard gadgets

- **My Tasks** gadget (workflow tasks assigned to me)
- **My Workflows** gadget (workflows started by me) with the columns:
  - doc,
  - step,
  - description,
  - user/group assigned,
  - delay.

Visualizing workflow state

There are two things that can be seen about the state of a workflow:

- the graph of current node and states,
- the audit (history) of workflow execution.

The graph is visible in the Workflow tab. Every node already done are shown with a green background (or something like this), every dead branch is grey, and others are just black.

The audit is shown in the History tab as a subtab, or may also be shown at the bottom of the Workflow tab.

Workflow progression

From the user's point of view, the progress in the workflow is done by going to a workflow task and clicking a button. This may require filling some form information first.

The workflow tasks are available in the user's dashboard, but should also be displayed to the user in the document's workflow tab (for 1-document workflows).

Modifying a running workflow

There may be various cases where the running workflow definition may need to be modified for a particular instance of workflow. Yet users won't design on a live workflow instance complex rules with conditional gates, etc... (at least in most of our customer cases). That's why we limit the ability to modify a workflow in this way:

- A user who is assigned a task (i.e., the workflow instance is stopped at a node of the graph) can add new steps, between the one he is in, and the one just after. This corresponds to cases where you know that someone would be able to better process the task, the review, etc. The available steps have been limited by the user who designed the workflow graph in Studio.
- Maybe a more complex security model should be added in the future to allow edition of any un-run step depending on persons, but this should come later.

If the user adds new nodes, they should be visible in the graphical representation of the workflow, in the same way that initial nodes are.

Saving a modified workflow

A user that modified a workflow dynamically may be allowed to save the workflow as a new workflow model. Suitable ACLs are set on the workflow model depending on how it should be shared.

XML export of a workflow model

It could be useful (although not a priority) to be able to export a workflow model (i.e., resulting from live modifications) as XML and then re-import it in Studio for modifications, permanent deployment, etc...

Migrating jBPM workflows

The code will be moved to an addon and kept identical.

New workflows based on Content Routing to provide serial reviews (with dynamic addition of steps) will be available by default to replace the old one.

Migrating Document Routing routes

They are kept available in the current model, just not used by default.

Migration of existing CMF Step models to the new node model, used in next version of CMF

We should declare existing “Distribute”, “Distribute with task”, etc... steps as node templates.