Workflows in Perl

Chris Winters

October 13, 2004

Where are we going?

- We'll do a brief intro of workflows
- Dive into Workflow with an example
- Along the way, talk about architecture

What we'll do?

- What is a workflow?
- Where have you seen them before?

But first, a disclaimer

- Workflows are both practical and academic
- Lots of products, lots of standards, lots of papers
- My experience is limited

But first, a disclaimer

- Workflows are both practical and academic
- Lots of products, lots of standards, lots of papers
- My experience is limited
- ... Some might say very limited

But first, a disclaimer

- Workflows are both practical and academic
- Lots of products, lots of standards, lots of papers
- My experience is limited
- ... Some might say very limited
- But: very open to suggestions, ideas, name-calling, etc.

What is it?

The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.

Okay, really. What is it?

- Curious feature: it's rarely defined
- Like "Content Management" – it can mean anything
- To bizfolk it's "Business Process Management"
- To developers it's "workflow"

Defined for our purposes

- Simple state machine
- Sequence of actions that defines a process

Defined for our purposes

- Simple state machine
- Sequence of actions that defines a process
- Yes, that's very vague too
- ...problem talking about generic frameworks...

Defined for our purposes (ctd)

At any point in our workflow we should be able to answer:

- Where am I right now?
- What data can I see?
- What can I do next?

Why a Workflow system?

- You probably use them all the time, just without the label
- Web applications are a great example
- State = web page; Transition = link/form

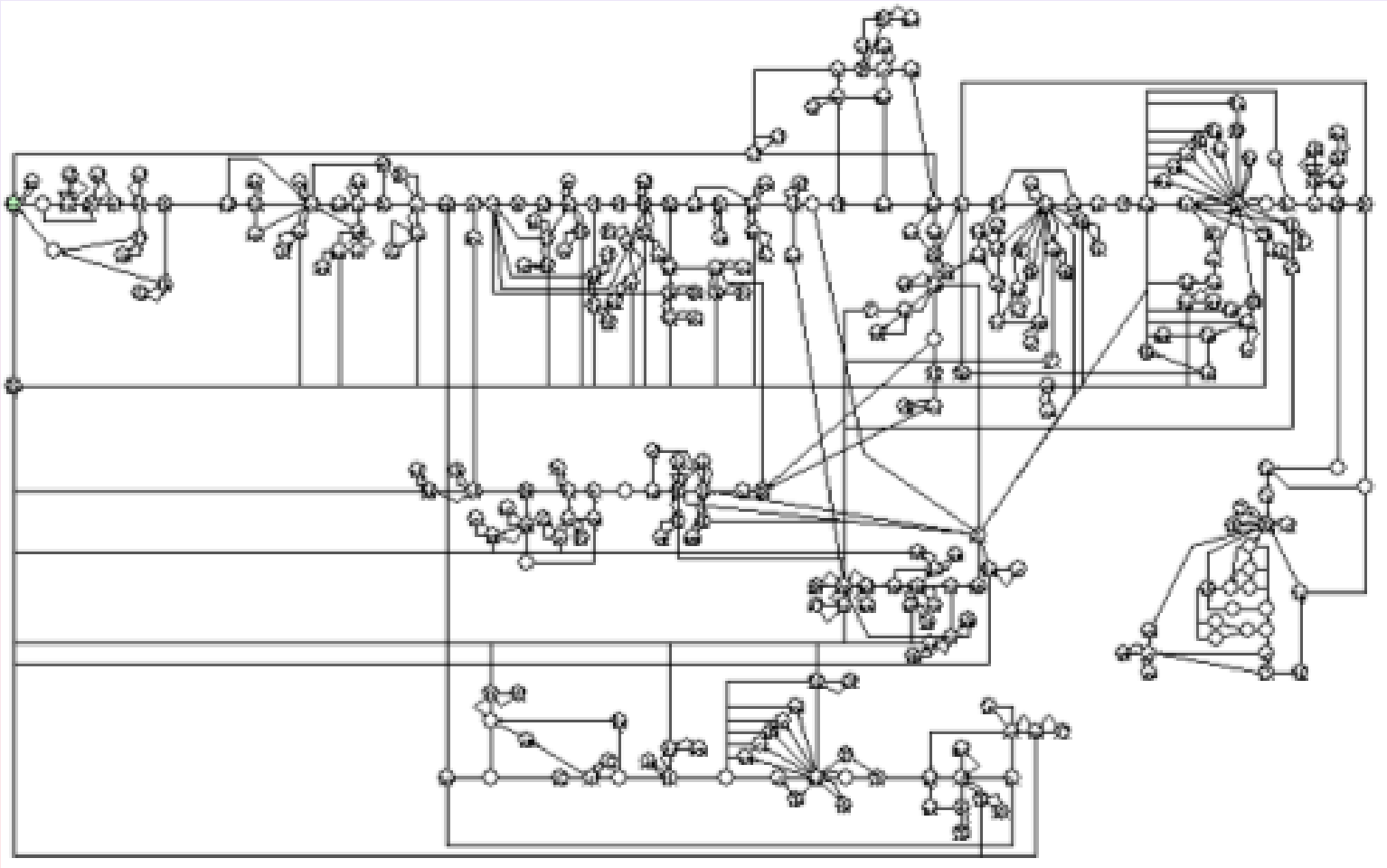
Why a Workflow system?

- You probably use them all the time, just without the label
- Web applications are a great example
- State = web page; Transition = link/form
- Place conditions on the menu links (transitions)
- ...based on who you are: admins see extra options
- ...based on where you're from: local weather

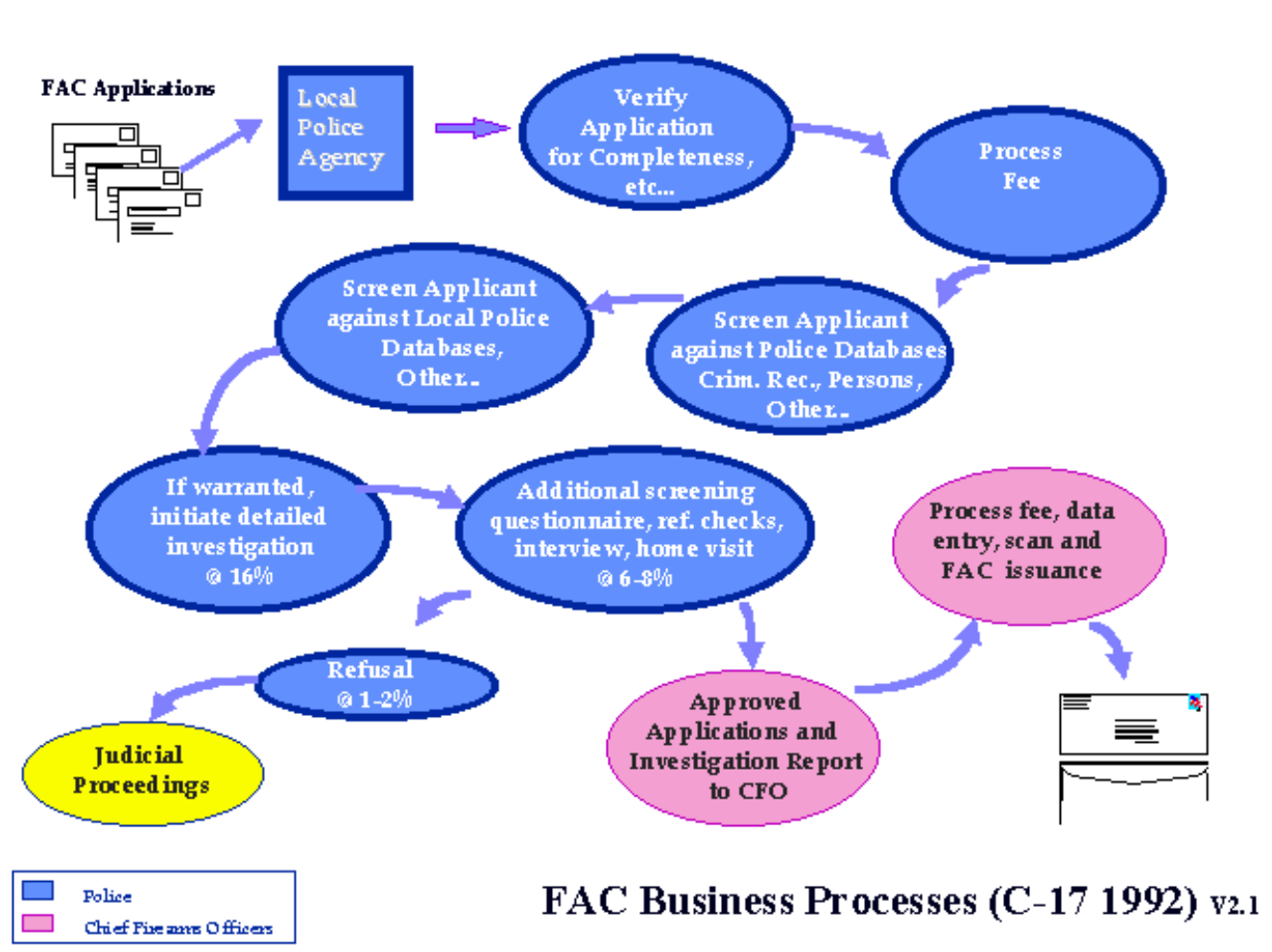
Workflow engines

- Workflow engine: not specific to any domain
- In theory: same engine for documents, purchasing, supply chain, etc.
- Lots of huge enterprise vendors and systems
- Sometimes tied to rule engines
- Tends glue different systems together
- Glue? Sound familiar?

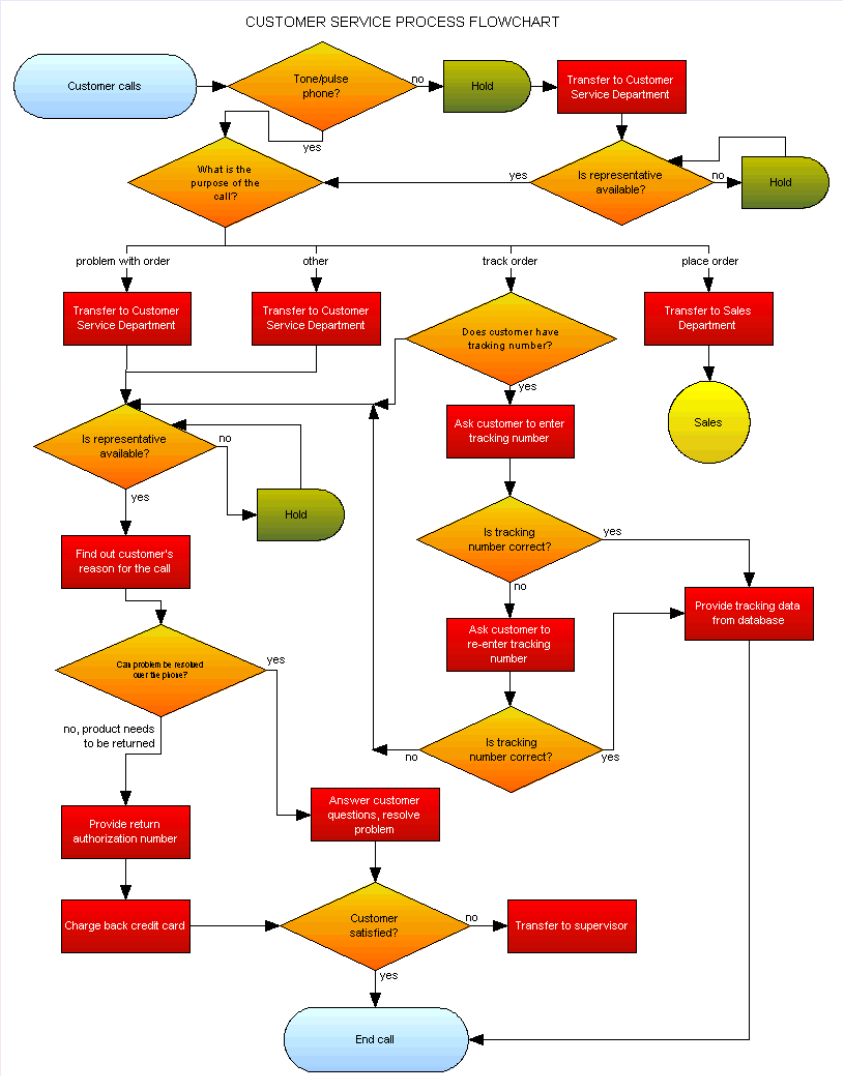
They can get very complicated...



And they're used everywhere...



And they're used everywhere...



Where have you seen them?

You see them every day:

- Receiving email through sendmail
- Submitting module to PAUSE
- Buying computer equipment from newegg
- Creating a bug with RT
- Submitting PO for new Powerbook
- Resolving issue with your gas company

Even if they're not explicit

- These systems might not have a 'workflow engine'
- Still doing the same job...
- ...may just be doing it the hard way

Lots of glue

- Workflows glue different processes together

Lots of glue

- Workflows glue different processes together
- Hey, Perl is a glue language

Lots of glue

- Workflows glue different processes together
- Hey, Perl is a glue language
- Natural fit!

Lots of glue

- Workflows glue different processes together
- Hey, Perl is a glue language
- Natural fit!
- Wrong!

Oh no, where's Perl?

There was no CPAN module for workflow systems!

- Lots of in-house workflows? Maybe...
- Fat lot of good that does us...

Enter: Workflow CPAN module

Embeddable, standalone workflow engine

- Comes with working example
- ...and tests! (not 100%...)
- ...and decent docs!

Simple state machine

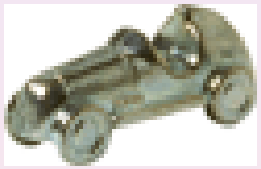
Simple state machines are easy:

State == "What's it doing now?"
Transition == "What can it do next?"

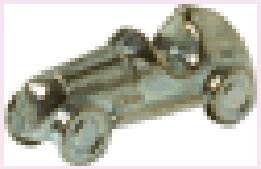
A real-world example

Board games and state machines

- Every square on the board is a state

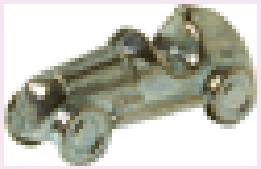


Board games and state machines



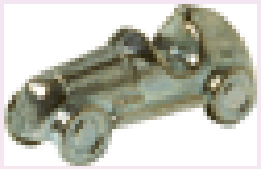
- Every square on the board is a state
- Every movement of your piece is a transition

Board games and state machines



- Every square on the board is a state
- Every movement of your piece is a transition
- ...can be instigated by die roll or command
- "Go to jail!" "Visit the Boardwalk..."

Board games and state machines



- Every square on the board is a state
- Every movement of your piece is a transition
- ...can be instigated by die roll or command
- "Go to jail!" "Visit the Boardwalk..."
- Your money and properties are application data

Workflow component lingo

- **Workflow** is your interface
- **States** are process stopping points
- **Actions** move between states
- **Validators** check data
- **Conditions** shield actions
- **Context** provides data
- ...all of them are just normal classes

How you create the workflow

- Configuration!

How you create the workflow

- Configuration!
- ...sorry, it's in XML (tried INI, honest)
- Very little code to instantiate

A set of XML files

One each for:

- workflow (required)
- actions (required)
- conditions (if necessary)
- validators (if necessary)
- persister (if necessary)

Naming the workflow

The workflow itself is just a few properties...

- The 'type' property is used to differentiate workflows
- Workflow 'type' analogous to object class

```
<workflow>
 <type>Issue</type>
 <description>Track issues</description>
 <persister>MyPersister</persister>
```

Using workflow 'type'

- With the 'type' we can instantiate new workflows or fetch existing ones

```
# Create a new one:
my $wf = Workflow::Factory
             ->create_workflow( 'Issue' );

# Get an existing one:
my $wf = Workflow::Factory
             ->fetch_workflow( 'Issue', 43 );
```

The factory is everywhere...

- Singleton pattern; can import `FACTORY`
- All persistence through factory: create, fetch, save
- Can ask the factory for conditions, validators
- See usage later in `App::Condition::IsCreator`

What are those components again?

- **Workflow** is your interface
- **States** are process stopping points
- **Actions** move between states
- **Validators** check data
- **Conditions** shield actions
- **Context** provides data

Workflow.push(@states)

Workflow is really just a bunch of states

```
<workflow>
  <state name="INITIAL" ... />
  <state name="ISSUE_CREATED" ... />
  <state name="ISSUE_IN_PROGRESS" ... />
  <state name="ISSUE_NEEDS_APPROVED" ... />
  <state name="ISSUE_CLOSED" ... />
```

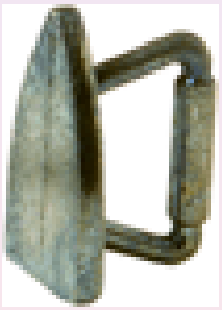
State.push(@actions)

A state is just a name and description, plus actions to execute.

```
<state name="INITIAL">
  <description>Initial issue state</description>
  <action name="create_issue"
          resulting_state="ISSUE_CREATED"/>
</state>
```

Actions move workflow between states

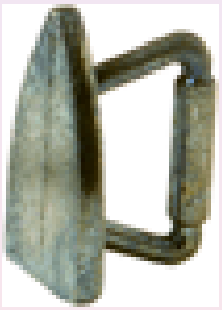
Just like our board game:



- move from square to square by rolling the die
- ...or by obeying the instructions on a card

Actions move workflow between states

Just like our board game:



- move from square to square by rolling the die
- ...or by obeying the instructions on a card
- Move from state to state by executing actions

Moving between states

Our example: trouble ticket system

- System starts when you create a ticket
- Developer decides to work on ticket
- ...executes 'Work on ticket' action
- ...workflow now at 'In Progress' state

Very simple

- The trouble ticket application is very, very simple
- Packaged with Workflow CPAN module (`eg/ticket/`)
- So you can play around quickly
- ...and break things quickly!

Multiple entry points

- You can manipulate workflows through web
- ...or through command-line shell interface
- Both reference same workflow configuration,
- ... same states, same actions, validators and conditions

Show the system...

Some architecture: Action

The main logic of your application is the Action

- One action can be used in multiple places
- Gets data from the workflow's context

Some architecture: Action

The main logic of your application is the Action

- One action can be used in multiple places
- Gets data from the workflow's context
- Can assume the data it wants is there!
- ...free to do the interesting stuff

Action the First: code

The Action is in three pieces; first: Perl class

- Subclasses `Workflow::Action`
- Must implement `execute()`
- Show a class...

Second: declare your action

- Tell the workflow framework about your action
- Done in action configuration (usually `workflow_action.xml`)
- Declare what data you need
- Hook in validators to ensure needed data are ok
- Show its declaration...

Third: hook into workflow

- Finally, make the action available in a state
- Done in workflow configuration (usually `workflow.xml`)
- Reference by name rather than class (readable)
- Tell your app what to do after executing action
- Hook in conditions: tell your app action is ok
- Show the hook...

Interrogating the workflow

Generic systems must be able to answer questions!

- Specialized workflow systems can hardcode more behavior
- Because we don't you have to do some more work
- But this can also make your system more robust...

Getting workflow actions

Want to know what actions you can run? Ask the workflow!

```
my $wf = Workflow::Factory
            ->fetch_workflow( 'Issue', 42 );
my @actions = $wf->get_current_actions;
print "You can do the following actions: ",
      join( ', ',  $wf->get_current_actions );
```

Actions to make a menu

So you can use this to create a generic menu (in TT):

```
[% FOREACH action = workflow.get_current_actions %]
( <a href="/execute?action=[% action %]">[% action %]</a> )
[% END %]
```

More reflection

Want to know required data for an action? Ask the workflow!

```
my @action_fields = $wf->get_action_fields( $action_name );
print "Data for action '$action_name':\n";
foreach my $field ( @action_fields ) {
    my @values = $field->get_possible_values;
    print join( "\n",
       "Field name: " . $field->name,
       "Type:       " . $field->type,
       "Required?   " . $field->is_required,
       "Values:     " . join( ', ', @values ),
       "Describe:   " . $field->description
   );
}
```

Feeding data to Action generically

- Since you can ask an action what data it needs
- ...which includes the name and data type...
- You can build a form/menu on the fly for user to input
- Which means your app doesn't have to!
- Show command-line example...
- See `Workflow::Action::InputField` for more...

More architecture: Validators

Similar scheme as other pieces:

- reference by name in action in state or action declaration
- declare in `workflow_validator.xml` (convention)
- full reference in separate file

Validators are simple

- Subclass `Workflow::Validator`
- Must implement `validate()`
- Normal objects, so they can have state
- Show example:
  `Workflow::Validator::MatchesDateFormat`

More architecture: Conditions

Similar scheme as other pieces:

- reference by name in action in state or action declaration
- declare in `workflow_condition.xml` (convention)
- can also inline for brevity

More architecture: Conditions

Similar scheme as other pieces:

- reference by name in action in state or action declaration
- declare in `workflow_condition.xml` (convention)
- can also inline for brevity
- ...version 0.10 added eval-able Perl conditions

Conditions are simple

- Subclass `Workflow::Condition`
- Must implement `evaluate()`
- Normal objects, so they can have state
- Show example: `App::Condition::IsCreator`
- Show inline example

More experimenting with the system...

- Nothing to see here

Just do it!

- Is this module perfect?
- Heck no!
- But it's better than none at all...

Resources

http://del.icio.us/cwinters/workflow

http://del.icio.us/tag/workflow

Q and A

- Thank you!
- Questions?