RUOTE
Business Processes in Ruby

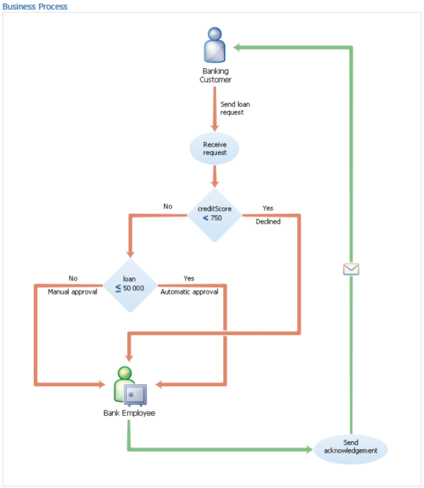
What is a BP?

A business process is a collection of related, structured activities or tasks that produce a specific service or product.

Business Process

It is often can be visualized with a flowchart as a sequence of activities with interleaving decision points.

Business Process



Ruote

Author: John Mettraux

Also:
Kenneth
Kalmer

Ruote is a Workflow Engine

Executes a Process

Expressions
control flow

Participants implement steps

Workitem passed to each participant

Storage maintains process state

Why

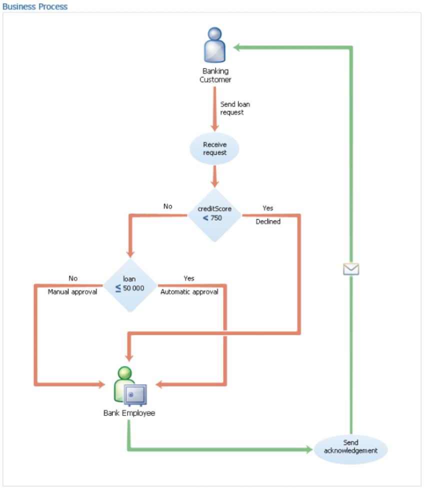
Business needs change quickly

Separate business logic from application

Accounting changed their mind!

Redeploy!

Business Process



Define Process

```ruby
process_def = Ruote.process_definition do
  _if '${credit_score} < 750' do
    alert_bank_employee( status: 'Declined' )
    _if '${amount} < 50000' do
      alert_bank_employee( status: 'Approved' )
      alert_bank_employee( status: 'Review' )
    end
  end
  send_acknowledgement
end

process_def.class # => Hash
```

Process Launch

```
dashboard.launch( process_def,
    credit_score: 770,
    amount: 100000 )
```

Processes Definitions

- ✓ Ruby
- ✓ JSON
- ✓ XML
- ✓ Radial

Store process outside of application

Expressions

- ✓ Sequence
- ✓ Concurrence
- ✓ Conditionals
- ✓ etc etc…

Expressions - Radial

```
if '${critical_error} == true'
  sequence
    alert_admins subject: 'Critical'
    shutdown
  concurrence
    alert_admins subject: 'Error'
    log_error
```

Expressions

```
cron '0 1 * * 1-5' # Every weekday @ 1am
  run_daily_accounts
```

Subprocesses

```
define 'handle_critical_error'
  sequence
    alert_admins subject: 'Critical'
    shutdown

define 'handle_error'
  concurrence
    alert_admins subject: 'Error'
    log_error
```

Subprocesses

```
if '${critical_error} == true'
  handle_critical_error
  handle_error
```

Advanced Expressions

- ✓ Implemented in Ruby
- ✓ Can add your own

Advanced Expressions

```
if_critical_error
  handle_critical_error
  handle_error
```

Bad Idea!

Participants

Perform work

Shared across processes

Participant Types

- ✓ Local - Class/Block
- ✓ Remote - HTTP/AMQP
- ✓ Storage - Offline

Class

```ruby
class SendAcknowledgement
  include Ruote::LocalParticipant

  def on_workitem( workitem )
    send_ack( workitem.fields[ :email_address ])
  end
end

dashboard.register_participant( :send_ack,
                                SendAcknowledgement )
```

Block

```
dashboard.register_participant( :send_ack) do |workitem|
    send_ack( workitem.fields[ :email_address ])
end
```

Storage Participant

- ✓ Stores the Workitem in Storage
- ✓ Will wait indefinitely
- ✓ Can be subclassed

Storage Participant

```ruby
class AlertBankEmployee < Ruote::StorageParticipant

  def on_workitem( workitem )
    alert_bank_employee( workitem.wfid )
    super
  end
end

dashboard.register_participant( :alert_bank_employee,
                                AlertBankEmployee )
```

Replying

```
w = dashboard.process( wfid ).workitems.first

dashboard.storage_participant.reply( w )
```

Storage

- ✓ File
- ✓ Sequel
- ✓ MongoDB
- ✓ Redis

Workers

- ✓ Storage allows multiple Workers
- ✓ Disconnect Workers from Dashboard

Workers

```
dashboard = Route::Dashboard.new( storage )
worker = Ruote::Worker.new( storage )
```

Workitem

- ✓ Passed to each participant
- ✓ Values assigned to fields
- ✓ Accessible in process

What to put in Workitem?

As much as possible!

Workitem

```ruby
process_def = Ruote.define do
  reviewer( task: 'spell_check' )
  reviewer( task: 'grammar' )
end

dashboard.launch( process_def, title: 'Pickaxe Book' )
```

Pros

Persistent

Separate business logic

Functional components

Cons

Initial Confusion

Testing (maybe)

Speed (maybe)

Resources

- ✓ ruote.rubyforge.org
- ✓ Ruote Google Group
- ✓ ruote on Freenode