We all know how frustrating experimenting can be.

That’s because experiments in distributed systems are:
- time-consuming
- difficult to do correctly
- complex and incomprehensible
- failure-prone
Automation of system administration

With tools like Chef and Puppet:
- a human factor is nearly removed
- systems are built from modules
- the configuration is reproducible

But reproducibility does not necessarily imply descriptiveness. It does not imply ease of understanding either.
Many tools to manage experiments exist:

- Expo
- g5k-campaign
- OMF
- Plush
- ... among many others

They are based on different paradigms.
Most of these tools use **bottom-up design**.

What about a **top-down** approach?

1. Start with high-level description of the experiment.
2. Implement low-level details.
3. Run the experiment.
4. Improve if necessary and reiterate.

There already exists an approach like this.

http://xpflow.gforge.inria.fr/
Business Process Management is about:

- understanding an organization
- modeling its processes as workflows
- executing processes and monitoring them
- improving organizational activities
- redesigning processes to make them:
  - cheaper
  - faster
  - less defective
Our solution, **XPFlow** is a merger of 3 domains:

- **Business Process Modeling** and **Management**
- **Scientific Workflows**
- it is a new **experimentation engine**
XPFlow workflows

Workflows (*processes*) in XPFlow are:
- based on BPM patterns (see Van Der Alst)
- written in a DSL
- orchestrate other processes and activities

Activities in XPFlow are:
- low-level, indivisible blocks of experiments
- written in Ruby

http://xpflow.gforge.inria.fr/
Workflows (\textit{processes}) in XPFlow are:
- based on BPM patterns (see Van Der Alst)
- written in a DSL
- orchestrate other processes and activities

\textbf{Activities} in XPFlow are:
- low-level, indivisible blocks of experiments
- written in Ruby
Workflows (processes) in XPFlow are:
- based on BPM patterns (see Van Der Alst)
- written in a DSL
- orchestrate other processes and activities

Activities in XPFlow are:
- low-level, indivisible blocks of experiments
- written in Ruby
The DSL for processes features different workflow patterns:
- running activities and other processes (run),
- running activities in order or in parallel (sequence, parallel),
- conditional expressions (if, switch)
- running sequential and parallel loops (loop, foreach, forall),
- error handling (try, checkpoint).

Some of them are taken directly from BPM.
Workflow patterns (example)

Activity A -- Activity B

Activity C (forall) |||

Activity D -- Activity E -- Activity F

Start event

Sequence

Parallel

Parallel loop

End event

http://xpflow.gforge.inria.fr/
Workflow patterns (example)

Start event

Activity A → Activity B → \( \forall \) Activity C → Activity D → Activity E → Activity F → End event

http://xpflow.gforge.inria.fr/
Workflow patterns (example)

Start event

Activity A → Activity B → Activity C (forall) → Activity D → Activity E → Activity F → End event

Sequence
Workflow patterns (example)

Start event

Activity A → Activity B → Activity C (forall) || Activity D → Activity E → Activity F

Sequence

Parallel

http://xpflow.gforge.inria.fr/
Workflow patterns (example)

Start event

Sequence

Parallel

Parallel loop

Activity A

Activity B

Activity D

Activity E

Activity F

Activity C (forall)

End event
Workflow patterns (example)
Workflow patterns (example, cont.)

```
process :workflow do |array|
  run :a
  run :b
  parallel do
    forall array do |x|
      run :c, x
    end
  sequence do
    run :d
    run :e
    run :f
  end
end
```

http://xpflow.gforge.inria.fr/
Workflow patterns (example, cont.)

Activity A -> Activity B

Activity C (forall)

Activity D -> Activity E -> Activity F

process : workflow do |array|
  run : a
  run : b
  parallel do
    forall array do |x|
      run : c, x
    end
  sequence do
    run : d
    run : e
    run : f
  end
end
Workflow patterns (example, cont.)

process :workflow do |array|
  run :a
  run :b
  parallel do
    forall array do |x|
      run :c, x
    end
  sequence do
    run :d
    run :e
    run :f
  end
end

http://xpflow.gforge.inria.fr/
Workflow patterns (example, cont.)

Activity A → Activity B → Activity C (forall) ||| Activity D → Activity E → Activity F

process : workflow do |array|
  run : a
  run : b
  parallel do
    forall array do |x|
      run : c, x
    end
  sequence do
    run : d
    run : e
    run : f
  end
end

http://xpflow.gforge.inria.fr/ XPFlow
Workflow patterns (example, cont.)

process :workflow do |array|
  run :a
  run :b
  parallel do
    forall array do |x|
      run :c, x
    end
  sequence do
    run :d
    run :e
    run :f
  end
end

http://xpflow.gforge.inria.fr/
Workflow patterns (example, cont.)

Activity A → Activity B → Activity C (forall) → Activity D → Activity E → Activity F

process :workflow do |array|
  run :a
  run :b
  parallel do
    forall array do |x|
      run :c, x
    end
  sequence do
    run :d
    run :e
    run :f
  end
end

http://xpflow.gforge.inria.fr/ XPFlow

11 / 20
Workflow patterns (example, cont.)

process : workflow do |array|
  run : a
  run : b
  parallel do
    forall array do |x|
      run : c, x
    end
  sequence do
    run : d
    run : e
    run : f
  end
end
#!/usr/bin/env xpflow

use :g5k

process :entry do
  job = g5k_get_avail :site => 'nancy', :jobid => var(:jid, :int)
  nodes = g5k_kadeploy(job, "wheezy-x64-nfs")
  checkpoint :cp
  r = execute_many nodes, "hostname"
  foreach r do |x|
    log stdout_of x
  end
end

main :entry

Assumes that xpflow is in your $PATH.
Error handling

XPFlow gives some means to cope with failures:

- **snapshotting:**
  - saves a state of an experiment for future use
  - shortens the development’s cycle

- **retry policy:**
  - retries a failed subprocess execution
  - improves reliability
  - allows to specify timeout

```ruby
process :snapshotting do
  run :long_deployment
  checkpoint :d
  run :experiment
end
```

```ruby
process :retrying do
  try :retry => 5 do
    run :tricky_activity
  end
end
```
Example of an experiment

Measure the **effective bisection bandwidth** of a switch.

1. Get names of all nodes connected to the switch.
2. Reserve the nodes.
3. Deploy Debian OS.
4. Install necessary software.
5. Compile and install *netgauge*.
6. Run the experiment.
7. Analyze results.
An experiment workflow

Few notes:

- each node must have some software installed
- each node must have *netgauge* installed ...
- ... but one node is enough to compile it
- one node must launch MPI application

We will introduce a *master* node and *slave* nodes.
An experiment workflow

Another observation: compilation can run in parallel with installation of software on the *slave* nodes.

http://xpflow.gforge.inria.fr/
This workflow describes our experiment.
The last thing to do is to express that in XPFlow.
An experiment workflow - DSL representation

process :exp do |site, switch|
  s = run g5k.switch, site, switch
  ns = run g5k.nodes, s
  r = run g5k.reserve_nodes,
      :nodes => ns, :time => '2h',
      :site => site, :type => :deploy
  master = (first_of ns)
  rest = (tail_of ns)
  run g5k.deploy,
      r, :env => 'squeeze-x64-nfs'
  checkpoint :deployed
  parallel :retry => true do
    forall rest do |slave|
      run :install_pkgs, slave
    end
  sequence do
    run :install_pkgs, master
    run :build_netgauge, master
    run :dist_netgauge,
        master, rest
  end
  end
  checkpoint :prepared
  output = run :netgauge, master, ns
  checkpoint :finished
  run :analysis, output, switch
end
An experiment workflow - DSL representation

```ruby
process :exp do |site, switch|
  s = run g5k.switch, site, switch
  ns = run g5k.nodes, s
  r = run g5k.reserve_nodes,
      :nodes => ns, :time => '2h',
      :site => site, :type => :deploy
  master = (first_of ns)
  rest = (tail_of ns)
  run g5k.deploy,
      r, :env => 'squeeze-x64-nfs'
  checkpoint :deployed
  parallel :retry => true do
    forall rest do |slave|
      run :install_pkgs, slave
    end
  end
  sequence do
    run :install_pkgs, master
    run :build_netgauge, master
    run :dist_netgauge,
        master, rest
  end
  checkpoint :prepared
  output = run :netgauge, master, ns
  checkpoint :finished
  run :analysis, output, switch
end
```

Activity :install_pkgs

```ruby
activity :install_pkgs do |node|
  log 'Installing packages on ', node
  run 'g5k.bash', node do
    aptget :update
    aptget :upgrade
    aptget :purge, 'mx'
  end
end
```

http://xpflow.gforge.inria.fr/
An experiment workflow - DSL representation

process :exp do |site, switch|
  s = run g5k.switch, site, switch
  ns = run g5k.nodes, s
  r = run g5k.reserve_nodes,
      :nodes => ns, :time => '2h',
      :site => site, :type => :deploy
  master = (first_of ns)
  rest = (tail_of ns)
  run g5k.deploy,
      r, :env => 'squeeze-x64-nfs'
  checkpoint :deployed
  parallel :retry => true do
    forall rest do |slave|
      run :install_pkgs, slave
    end
  sequence do
    run :install_pkgs, master
    run :build_netgauge, master
    run :dist_netgauge, master, rest
  end
  checkpoint :prepared
  output = run :netgauge, master, ns
  checkpoint :finished
  run :analysis, output, switch
end

Activity :build_netgauge

activity :build_netgauge do |master|
  log "Building netgauge on #{master}"
  run 'g5k.copy', NETGAUGE, master, '~'
  run 'g5k.bash', master do
    build_tarball NETGAUGE, PATH
  end
  log "Build finished."
end
An experiment workflow - DSL representation

process :exp do |site, switch|
  s = run g5k.switch, site, switch
  ns = run g5k.nodes, s
  r = run g5k.reserve_nodes,
      :nodes => ns, :time => '2h',
      :site => site, :type => :deploy
  master = (first_of ns)
  rest = (tail_of ns)
  run g5k.deploy,
      r, :env => 'squeeze-x64-nfs'
  checkpoint :deployed
  parallel :retry => true do
    forall rest do |slave|
      run :install_pkgs, slave
    end
  sequence do
    run :install_pkgs, master
    run :build_netgauge, master
    run :dist_netgauge, master, rest
  end
  checkpoint :prepared
  output = run :netgauge, master, ns
  checkpoint :finished
  run :analysis, output, switch
end

Activity :dist_netgauge

activity :dist_netgauge do |m, s|
  master, slaves = m, s
  run 'g5k.dist_keys', master, slaves
  run 'g5k.bash', master do
    distribute BINARY,
    DEST, 'localhost', slaves
  end
end
An experiment workflow - DSL representation

```
process :exp do |site, switch|
  s = run g5k.switch, site, switch
  ns = run g5k.nodes, s
  r = run g5k.reserve_nodes,
      :nodes => ns, :time => '2h',
      :site => site, :type => :deploy
  master = (first_of ns)
  rest = (tail_of ns)
  run g5k.deploy,
      r, :env => 'squeeze-x64-nfs'
  checkpoint :deployed
  parallel :retry => true do
    forall rest do |slave|
      run :install_pkgs, slave
    end
  sequence do
    run :install_pkgs, master
    run :build_netgauge, master
    run :dist_netgauge,
        master, rest
  end
  checkpoint :prepared
  output = run :netgauge, master, ns
  checkpoint :finished
  run :analysis, output, switch
end
```

Activity :netgauge

```
activity :netgauge do |master, nodes|
  log "Running experiment..."
  out = run 'g5k.bash', master do
    cd PATH
    mpirun nodes, "./netgauge"
  end
  log "Experiment done."
end
```
Running the experiment

The experiment runs on Grid’5000 frontend or on your local machine.

The execution is monitored and errors reported if necessary.
Each activity is monitored during its execution.
Notice that build_netgauge:1 runs in parallel with install_pkgs:*.
In these few slides we presented XPFlow.

Current features include:
- improved descriptiveness
- modularity and flexibility
- monitoring and support for common patterns
- robustness in case of failures
- scalability of experiments
- integration with Grid’5000

More things to come:
- better user interface
- improved checkpointing
- support for provenance
- easier result management
- modules

http://xpflow.gforge.inria.fr/
Interested?

Visit

http://xpflow.gforge.inria.fr