Developing with DREAM Platform

Nexedi 2015-03-26
Agenda

1. Install DREAM Runner
2. Edit DREAM platform source code
3. Run DREAM platform unit tests
4. Accessing DREAM GUI
5. Introduction to JSON format
6. Configuring Graph Editor
7. Configuring Application configuration, adding a spreadsheet tab and a new gadget
8. Configuring Knowledge extraction plugins
9. Cloud Execution of ACO
Install DREAM Runner

Easy way: use vifib hosting

1. Create an account on https://www.slapos.org/
Install DREAM Runner

Easy way: use vifib.com hosting

1. Create an account on https://www.slapos.org/
2. Request New Service
Install DREAM Runner

Easy way: use vifib.com hosting

1. Create an account on [https://www.slapos.org/](https://www.slapos.org/)
2. Request New Service
3. Choose “SlapOS web runner”
Install DREAM Runner

Easy way: use vifib.com hosting

1. Create an account on https://www.slapos.org/
2. Request New Service
3. Choose “SlapOS web runner”
4. Choose latest version

<table>
<thead>
<tr>
<th>SlapOS Web Runner (Outdated)</th>
<th>slapos-0.244</th>
<th>Production and Commercial version of SlapOS Web Runner</th>
</tr>
</thead>
<tbody>
<tr>
<td>SlapOS Web Runner</td>
<td>slapos-0.254</td>
<td>Production and Commercial version of SlapOS Web Runner</td>
</tr>
<tr>
<td>SlapOS Web Runner (Outdated)</td>
<td>slapos-0.252.1</td>
<td>Production and Commercial version of SlapOS Web Runner</td>
</tr>
<tr>
<td>SlapOS Web Runner</td>
<td>slapos-0.258</td>
<td>Production and Commercial version of SlapOS Web Runner</td>
</tr>
</tbody>
</table>
Install DREAM Runner

Easy way: use vifib.com hosting

1. Create an account on [https://www.slapos.org/](https://www.slapos.org/)
2. Request New Service
3. Choose “SlapOS web runner”
4. Choose latest version
5. Set Service Title and XML Parameter
Install DREAM Runner

Easy way: use vifib.com hosting

1. Create an account on https://www.slapos.org/
2. Request New Service
3. Choose “SlapOS web runner”
4. Choose latest version
5. Set Service Title and XML Parameter
   - slapos-software: the software to use, available in slapos git repository, see http://git.erp5.org/gitweb/slapos.git/tree/HEAD:/software?js=1
   - slapos-reference: The git branch from slapos repository
   - auto-deploy: Auto deploy the software
   - user-authorized-keys: Your ssh public key, if you want to use ssh
   - custom-frontend-backend-url: This will generate an IPv4 frontend for you if you cannot access it through IPv6.
Install DREAM Runner

Easy way: use vifib.com hosting

1. Create an account on https://www.slapos.org/
2. Request New Service
3. Choose “SlapOS web runner”
4. Choose latest version
5. Set Service Title and XML Parameter
6. Wait for a while and connection parameter will appear
Install DREAM Runner

Easy way: use vifib.com hosting

1. Create an account on https://www.slapos.org/
2. Request New Service
3. Choose “SlapOS web runner”
4. Choose latest version
5. Set Service Title and XML Parameter
6. Wait for a while and connection parameter will appear
7. Click on **URL** parameter
Install DREAM Runner

Easy way: use vifib.com hosting

1. Create an account on https://www.slapos.org/
2. Request New Service
3. Choose “SlapOS web runner”
4. Choose latest version
5. Set Service Title and XML Parameter
6. Wait for a while and connection parameter will appear
7. Click on URL parameter
8. **Pay invoices every month**
Install DREAM Runner

… or Install SlapOS in your IT infrastructure.
Follow the tutorials from:
Install DREAM Runner

Before start, configure monitoring from “Monitoring” link on the slapos page.

1. Set monitoring password

This is the monitoring interface

Please set your password for later access

Password*: Set your password
Verify Password*: Verify password

Access
Install DREAM Runner

Before start, configure monitoring from “Monitoring” link on the slapos page.

1. Set monitoring password
2. Set recovery code

**Monitoring**

<table>
<thead>
<tr>
<th>MONITOR-PUBLIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>rssfeed.html</td>
</tr>
<tr>
<td>MONITORING</td>
</tr>
<tr>
<td>status-history.cgi</td>
</tr>
<tr>
<td>status.cgi</td>
</tr>
<tr>
<td>ZERO-KNOWLEDGE</td>
</tr>
<tr>
<td>settings.cgi</td>
</tr>
<tr>
<td>monitor-password.cgi</td>
</tr>
<tr>
<td>cors-domain.cgi</td>
</tr>
</tbody>
</table>

**Values that can be defined:**

- recovery-code: uamdntrxf
- status-history-length: 5
- shell-password: flcymqwh

**Other values:**
Install DREAM Runner

Upon first login, configure your account
Set your name, email, login and password
Enter Password Recovery Code from monitoring interface
Install DREAM Runner

Click play button to compile software and create instance
Install DREAM Runner

Wait for building state and running state to be Complete
Install DREAM Runner

Check your processes are running in “Process” tab. From this screen, you can also restart processes.

This tab shows all processes generated by slapgrid for your application. You can click on the process name to display log.

<table>
<thead>
<tr>
<th>Partition and Process name</th>
<th>Status</th>
<th>Process PID</th>
<th>UpTime</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>slappart0:dream_grunt_watch-on-watch</td>
<td>RUNNING</td>
<td>12567</td>
<td>0:04:52</td>
<td>Restart</td>
</tr>
<tr>
<td>slappart0:dream_platform-on-watch</td>
<td>RUNNING</td>
<td>12538</td>
<td>0:04:52</td>
<td>Restart</td>
</tr>
<tr>
<td>slappart0:dream_simulation</td>
<td>EXITED</td>
<td>25</td>
<td>PM</td>
<td>Restart</td>
</tr>
<tr>
<td>slappart0:dream_test_suite</td>
<td>RUNNING</td>
<td>18925</td>
<td>0:00:53</td>
<td>Restart</td>
</tr>
</tbody>
</table>

[Refresh Status] [Stop all process]
Edit DREAM platform source code

Editor tab let you edit files:
Edit DREAM platform source code

To access DREAM source code, first switch from “This project” to “Working dir”
Edit DREAM platform source code

DREAM source code is in:
  software
    (the hash of the software)
    parts
    dream-repository.git

This is a git working copy of http://git.erp5.org/gitweb/dream-repository.git/
Edit DREAM platform source code

Tips: add this folder for favourites, it will be available from the favourites menu
Edit DREAM platform source code

ManPy code can be edited:

```python
# Copyright 2013 University of Limerick
#
# This file is part of DREAM.
#
# DREAM is free software: you can redistribute it and/or modify
# it under the terms of the GNU Lesser General Public License
# the Free Software Foundation, either version 3 of the License,
# or (at your option) any later version.
#
# DREAM is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty
# of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
# the GNU Lesser General Public License for more details.
#
# You should have received a copy of the GNU Lesser General
# Public License along with DREAM.  If not, see <http://www.gnu.org/licenses/>
#
Created on 8 Nov 2012
@Author: George

Models a machine that can also have failures
```

Running DREAM test suite

After making changes, run tests to validate your changes

<table>
<thead>
<tr>
<th>Process and Process name</th>
<th>Status</th>
<th>Process PID</th>
<th>UpTime</th>
<th>Monitoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>slappart0:dream_grunt_watch-on-watch</td>
<td>RUNNING</td>
<td>12567</td>
<td>0:23:24</td>
<td>Restart</td>
</tr>
<tr>
<td>slappart0:dream_platform-on-watch</td>
<td>RUNNING</td>
<td>12538</td>
<td>0:23:24</td>
<td>Restart</td>
</tr>
<tr>
<td>slappart0:dream_simulation</td>
<td>EXITED</td>
<td>25</td>
<td>PM</td>
<td>Restart</td>
</tr>
<tr>
<td>slappart0:dream_test_suite</td>
<td>EXITED</td>
<td>25</td>
<td></td>
<td>Restart</td>
</tr>
</tbody>
</table>

This tab shows all processes generated by slapgrid for your application. You can click on the process name to display log.
Running DREAM test suite

After making changes, run tests to validate your changes

<table>
<thead>
<tr>
<th>Process</th>
<th>Connection Information</th>
<th>Parameters</th>
<th>Partitions Content</th>
<th>Monitoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>slappart0:dream_grunt_watch-on-watch</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slappart0:dream_platform-on-watch</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slappart0:dream_simulation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slappart0:dream_test_suite</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This tab show all process generated by slapgrid for your application. You can click on the process name to display log.

2. Click on the process name to view the output
Accessing DREAM GUI

The GUI instance can be accessed from **custom frontend URL** in SlapOS parameter page.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>backend_url</td>
<td>https://[2001:67c:1254:e:1c::b4c0]:50005</td>
</tr>
<tr>
<td>2_info</td>
<td>In order to set up your account, get the recovery-code from the monitoring interface. Before read the notification on monitor_info.</td>
</tr>
<tr>
<td>ssh_command</td>
<td>ssh 2001:67c:1254:e:1c::b4c0 -p 22222</td>
</tr>
<tr>
<td>url</td>
<td><a href="https://softins59074.host.vifib.net">https://softins59074.host.vifib.net</a></td>
</tr>
<tr>
<td>public_url</td>
<td><a href="https://softins59072.host.vifib.net/public/">https://softins59072.host.vifib.net/public/</a></td>
</tr>
<tr>
<td>webdav_url</td>
<td><a href="https://softins59072.host.vifib.net/share/">https://softins59072.host.vifib.net/share/</a></td>
</tr>
<tr>
<td>git_public_url</td>
<td>https://[2001:67c:1254:e:1c::b4c0]:9684/git-public/</td>
</tr>
<tr>
<td>git_private_url</td>
<td>https://[2001:67c:1254:e:1c::b4c0]:9684/git/</td>
</tr>
<tr>
<td>custom-frontend-url</td>
<td><a href="https://softins59073.host.vifib.net">https://softins59073.host.vifib.net</a></td>
</tr>
<tr>
<td>access_url</td>
<td><a href="https://softins59074.host.vifib.net/login">https://softins59074.host.vifib.net/login</a></td>
</tr>
<tr>
<td>monitor_backend_url</td>
<td>https://[2001:67c:1254:e:1c::b4c0]:9684</td>
</tr>
<tr>
<td>1_info</td>
<td>On your first run, Use &quot;access_url&quot; to setup you account. Then you can use both &quot;url” or &quot;access_url&quot;. Or &quot;backend_url&quot; if you want to use ipv6. Set up your account in the webrunner in order to use webdav, and being able to clone your git repositories from the runner.</td>
</tr>
<tr>
<td>monitor_url</td>
<td><a href="https://softins59072.host.vifib.net">https://softins59072.host.vifib.net</a></td>
</tr>
</tbody>
</table>
Accessing DREAM GUI

Load a model, for example https://raw.githubusercontent.com/nexedi/dream/master/dream/simulation/Examples/GUI_instances/DefaultAllInOneEmpty.json
Accessing DREAM GUI

Create a simple model
Accessing DREAM GUI

Edit model properties
Accessing DREAM GUI

Running simulation

<table>
<thead>
<tr>
<th>Machine Shifts Spreadsheet</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Repairman Shifts Spreadsheet</td>
<td></td>
</tr>
</tbody>
</table>

Run Simulation

Confidence level for statistical analysis of stochastic experiments

0.95

The day the experiment starts, in YYYY/MM/DD HH:MM:SS format

2015/01/01 09:00:00

The URL for knowledge extraction to access its data for example http://git.erp5.org/gitweb/dream.git/blob_plain/HEAD:

dream...
Accessing DREAM GUI

Visualise results

Exit Statistics

<table>
<thead>
<tr>
<th>Exit Id</th>
<th>Throughput</th>
<th>Takt Time</th>
<th>Lifespan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dream.Exit1</td>
<td>1000</td>
<td>0.099899999999986</td>
<td>0.0812500000000017</td>
</tr>
</tbody>
</table>
Introduction to JSON format

Simple Representation of a Directed Graph

```
{
    "node": {
        "S1": {
            "name": "S1"
        },
        "M1": {
            "name": "M1"
        },
        "E1": {
            "name": "E1"
        }
    },
    "edge": {}
}
```
Introduction to JSON format

Nodes connected by edges

```json
{
  "node": {
    "S1": {
      "name": "S1"
    },
    "M1": {
      "name": "M1"
    },
    "E1": {
      "name": "E1"
    }
  },
  "edge": {
    "edge1": {
      "source": "S1",
      "destination": "M1"
    },
    "edge2": {
      "source": "M1",
      "destination": "E1"
    }
  }
}
```
Introduction to JSON format

Properties of nodes

```
{
    "node": {
        "S1": {
            "name": "Parts",
            "_class": "Dream.Source",
            "interarrivalTime": {
                "distributionType": "Fixed",
                "mean": 0.5
            },
            "entity": "Dream.Part"
        },
        "M1": {
            "name": "M1"
        },
        "E1": {
            "name": "E1"
        }
    },
    "edge": {
        "edge1": {
            "source": "S1",
            "destination": "M1"
        },
        "edge2": {
            "source": "M1",
            "destination": "E1"
        }
    }
}
```
Introduction to JSON format

- What are the editable properties?
- Which value type (text, number)?
- What possible values?
- What default value?
Introduction to JSON format

Properties are defined as JSON Schema

```
"class_definition": {
    "Dream.Source": {
        "properties": {
            "id": {
                "type": "string",
                "default": "S",
                "required": true,
                "name": "ID",
                "description": "The ID of this source"
            },
            "name": {
                "type": "string",
                "default": "Source"
            },
            "interarrivalTime": {
                "type": "object",
                "properties": {
                    "distributionType": {
                        "type": "string",
                        "default": "Fixed",
                        "enum": [ "Fixed", "Normal", "Exponential" ]
                    },
                    "mean": {
                        "type": "number",
                        "default": 0.5
                    }
                }
            }
        }
    }
}
```

ID
S1

Name
Parts

Interarrival Time

Distribution Type

Fixed

Mean
0.5

Standard Deviation

Minimum Value

Maximum Value

Entity Class
Dream.Part
Introduction to JSON format

class_definition: {
  Dream.Source: {
    properties: {
      id: {
        type: "string",
        default: "S",
        required: true,
        name: "ID",
        description: "The ID of this source"
      },
      name: {
        type: "string",
        default: "Source"
      },
      interarrivalTime: {
        type: "object",
        properties: {
          distributionType: {
            type: "string",
            default: "Fixed",
            enum: ["Fixed", "Normal", "Exponential"]
          },
          mean: {
            type: "number",
            default: 0.5
          }
        }
      }
    }
  },
  name: {
    type: "string",
    default: "Source"
  },
  interarrivalTime: {
    type: "object",
    properties: {
      distributionType: {
        type: "string",
        default: "Fixed",
        enum: ["Fixed", "Normal", "Exponential"]
      },
      mean: {
        type: "number",
        default: 0.5
      }
    }
  }
}

ID
S1
Name
Parts
Interarrival Time
Distribution Type
Fixed
Mean
0.5
Standard Deviation
Minimum Value
Maximum Value
Entity Class
Dream.Part
Introduction to JSON format

```
"class_definition": {
    "Dream.Source": {
        "properties": {
            "id": {
                "type": "string",
                "default": "S",
                "required": true,
                "name": "ID",
                "description": "The ID of this source"
            },
            "name": {
                "type": "string",
                "default": "Source"
            },
            "interarrivalTime": {
                "type": "object",
                "properties": {
                    "distributionType": {
                        "type": "string",
                        "enum": ["Fixed", "Normal", "Exponential"],
                        "default": "Fixed"
                    },
                    "mean": {
                        "type": "number",
                        "default": 0.5
                    }
                }
            }
        }
    }
},
"name": {
    "type": "string",
    "default": "Source"
}
```

Introduction to JSON format

```
"class_definition": {
    "Dream.Source": {
        "properties": {
            "id": {
                "type": "string",
                "default": "S",
                "required": true,
                "name": "ID",
                "description": "The ID of this source"
            },
            "name": {
                "type": "string",
                "default": "Source"
            },
            "interarrivalTime": {
                "type": "object",
                "properties": {
                    "distributionType": {
                        "type": "string",
                        "default": "Fixed",
                        "enum": ["Fixed", "Normal", "Exponential"],
                    },
                    "mean": {
                        "type": "number",
                        "default": 0.5
                    }
                }
            }
        }
    }
}
```
Introduction to JSON format

General purpose tabular inputs in the same JSON

<table>
<thead>
<tr>
<th>Order ID</th>
<th>Due Date</th>
<th>Priority</th>
<th>Project Manager</th>
<th>Part</th>
<th>Part Type</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order 1</td>
<td>2014/03/15</td>
<td>1</td>
<td>PM1</td>
<td>Design</td>
<td>Design</td>
<td>CAD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Part1</td>
<td>Basic</td>
<td>CAM-MILL-EDM-MILL-MASS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Part2</td>
<td>Basic</td>
<td>CAM-MILL-EDM-MILL-MASS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Assemble</td>
<td>Mould</td>
<td>MASS-IM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2-12</td>
</tr>
<tr>
<td>Order 2</td>
<td>2014/03/14</td>
<td>1</td>
<td>PM1</td>
<td>Design</td>
<td>Design</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Part1</td>
<td>Basic</td>
<td>CAM-MILL-EDM-MILL-MASS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Part2</td>
<td>Basic</td>
<td>CAM-MILL-EDM-MASS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Assemble</td>
<td>Mould</td>
<td>MASS-IM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8-4-2-8-0</td>
</tr>
<tr>
<td>Order 3</td>
<td>2014/03/15</td>
<td>1</td>
<td>PM1</td>
<td>Design</td>
<td>Design</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Part1</td>
<td>Basic</td>
<td>CAM-MILL-EDM-MASS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Part2</td>
<td>Basic</td>
<td>CAM-MILL-EDM-MASS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Assemble</td>
<td>Mould</td>
<td>MASS-IM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7-15-8-8-0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1-3</td>
</tr>
</tbody>
</table>
Introduction to JSON format

Outputs - results of simulation - are also in the same JSON
Introducing Edit Configuration tab

Live edition of JSON
Introducing Edit Configuration tab

Live edition of JSON

Features non obtrusive assistance with syntax errors in the JSON

```
{  
  "application_configuration": {  
    "general": {  
      "properties": {  
        "expected": "STRING",  
        "got": "undefined"  
      }  
    }  
  }  
}
```

`Parse error on line 10:  
... },           ERROR  
-------------^  
Expecting 'STRING', got 'undefined'

```
"currentDate": {  
  "default": "2014/10/01 09:00:00",  
  "description": "The day the experiment st  
  "title": "SimulationStartTime".
```
Graph editor: adding a property on node

On Edit configuration tab:
Locate the json schema definition of Dream.Queue in class_definition

```json
{
  "application_configuration": { },
  "class_definition": {
    "Dream.Edge": { },
    "Dream.EventGenerator": { },
    "Dream.Exit": { },
    "Dream.Machine": { },
    "Dream.Queue": {
      "_class": "node",
      "allOf": [
        {
          "$ref": "#/node"
        }
      ],
      "properties": {
        "capacity": {
          "$ref": "#/definitions/_capacity",
          "required": true
        }
      },
      "id": {
        "default": "Q",
        "type": "string"
      }
    }
  }
}
```
Graph editor: adding a property on node

On Edit configuration tab:
Add a new property, in this example it is called “Sample property”
Graph editor: adding a property on a node

On “Production Line” tab, we have a new property when editing a Dream.Queue.

Set value “user value” and save
Go back to “Edit configuration” tab, we can see that the sample_property is saved as a property on the node.
Application configuration: adding a tab

On Edit configuration tab:
Locate the input action definition in application_definition

input is for input action
output is for results visualisation
Application configuration: adding a tab

Add an action:

*gadget*: The gadget to use
*title*: The name to display on the tab
*type*: Always object_view for now
*configuration*: Gadget configuration, passed to gadget constructor.
  
  *columns*: columns definition
  
  *input_id*: key for this input
Application configuration: adding a tab

The new tab appears. Set some values

<table>
<thead>
<tr>
<th>Sample Spreadsheet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Production Line</td>
</tr>
<tr>
<td>Manage document</td>
</tr>
<tr>
<td>Results</td>
</tr>
<tr>
<td>WIP Spreadsheet</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sample Column</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>value 1</td>
<td></td>
</tr>
<tr>
<td>value 2</td>
<td></td>
</tr>
<tr>
<td>value 3</td>
<td></td>
</tr>
</tbody>
</table>
The values are saved in the JSON, as input / (the input_id configured)
Application configuration: adding a tab

To add a completely new gadget, locate the gadget folder in dream-repository.
git/dream/platform/src/dream
Application configuration: adding a tab

- Clone existing gadget:
  - html file, usually Input_viewXXXGadget.html
  - javascript file, usually Input_viewXXXGadget.js
- Make sure html file reference the javascript properly
- Debugging tips:
  - code you edit is in dream/platform/src/
  - grunt compiles code in dream/platform/static
  - if syntax error, files in static are not updated!
  - static files have different lines numbers
- use console.log (better with firebug)
- [http://learn.renderjs.org/docs/index.html](http://learn.renderjs.org/docs/index.html)
Pre/Post Processing plugins

JSON data goes through:
- pre processing plugins
- simulation engine
- post processing plugins
Pre/Post Processing plugins

Plugins configured in JSON, under *application_configuration*

```json
{
  "application_configuration": {
    "general": {},
    "input": {},
    "output": {},
    "post_processing": {
      "plugin_list": [
        {
          "_class": "dream.plugins.DefaultTabularExit.DefaultTabularExit"
        },
        {
          "_class": "dream.plugins.PostProcessStationUtilization.PostProcessStationUtilization",
          "family": "Server",
          "output_id": "station_utilization"
        },
        {
          "output_id": "queue_statistics"
        }
      ]
    },
    "pre_processing": {
      "description": ""
    }
  }
}
```

Python class for plugin

Parameters for the plugin
Pre/Post Processing plugins

Plugins are python class, interface is postprocess(self, data) or preprocess(self, data)

```python
from dream.plugins import plugin

class PostProcessStationUtilization(plugin.OutputPreparationPlugin):
    ''' Output the station utilization metrics in a format compatible with DREAM
    '''

    def postprocess(self, data):
        result = data['result']['result_list'][-1]

        ticks = []
        working_data = []
        waiting_data = []
        failure_data = []
        blockage_data = []
        setup_data = []

        options = {
            "xaxis": {
                "minTickSize": 1,
                "ticks": ticks
            },
            "yaxis": {
                "max": 100
            },
            "series": {
                "show": True,
                "barWidth": 0.8,
                "align": "center"
            }
        }
```

FoF.NMP.2012-6
Collaborative Project
Grant Agreement: 314364

DREAM
Pre/Post Processing plugins

Plugin configuration defined in json is in self.configuration_dict

```python
out = result[self.configuration_dict['output_id']] = {
    "series": series,
    "options": options
}

i = 0
for obj in result['elementList']:
    if obj.get('family') == self.configuration_dict.get('family'):
        if obj['results']['working_ratio']:
            working_data.append((i, obj['results']['working_ratio'][0]))
        if obj['results']['waiting_ratio']:
            waiting_data.append((i, obj['results']['waiting_ratio'][0]))
        if obj['results']['failure_ratio']:
            failure_data.append((i, obj['results']['failure_ratio'][0]))
        if obj['results']['blockage_ratio']:
            blockage_data.append((i, obj['results']['blockage_ratio'][0]))
        if obj['results']['setup_ratio']:
            setup_data.append((i, obj['results']['setup_ratio'][0]))
        ticks.append((i, obj.get('name', obj['id'][0])))
        i += 1

return data
```
Pre/Post Processing plugins

Plugins debugging tips:

- Use existing Knowledge Extraction Tool objects (from dream/KnowledgeExtraction/ folder)
- Use dream.plugin.TimeSupport to convert from and to simulation clock time
- Log using self.logger.info (standard python logger https://docs.python.org/2/library/logging.html)
- Use dream.plugin.Debug in the chain to dump the json
- Write unit tests (examples in dream/tests/testGUIModels.py)
Accessing the application logs

Add a file in log panel (1/2)
Accessing the application logs

Add a file in log panel (2/2)
Cloud Execution

Simulation Web Server

Web Browser
DREAM UI

Simulation Slave: Sim Engine

Simulation Slave: Sim Engine

Simulation Slave: Sim Engine

Simulation Slave: Sim Engine

Distributor

Scenarios 1..N

Simulation Web Server:
1. Build a list of scenarios to be evaluated
2. Send this list to the distributor
3. Wait for distributor to have finished.
4. Return the best scenarios when all are evaluated

Scenario 1

Scenario 2

Scenario N
Cloud Execution

- Note the custom frontend URL for later reference
- Install some DREAM simulation nodes [http://git.erp5.org/gitweb/slapos.git/blob_plain/refs/heads/dream:/software/testnode/software.cfg](http://git.erp5.org/gitweb/slapos.git/blob_plain/refs/heads/dream:/software/testnode/software.cfg) set the distributor URL as “test-suite-master-url” parameter. Or using the following XML Parameter for web runner:

```xml
<instance>
  <parameter id="parameter-test-node-title">Simulation Node 1</parameter>
  <parameter id="parameter-test-suite-master-url">https://login:password@softinstXXX.host.vifib.net/erp5/portal_task_distribution/dream_distributor</parameter>
  <parameter id="slapos-software">software/erp5testnode</parameter>
  <parameter id="slapos-reference">dream</parameter>
</instance>
```
Cloud Execution

Be sure to use dream.plugin.ACO.ACO as processing plugin class in your model
### Cloud Execution

Be sure to specify the distributor URL in general properties

<table>
<thead>
<tr>
<th>Seed for random number generator</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of generations</td>
<td>1</td>
</tr>
<tr>
<td>Number of ants per generation</td>
<td>20</td>
</tr>
<tr>
<td>Number of solutions</td>
<td>10</td>
</tr>
<tr>
<td>Distributor URL</td>
<td>http://(custom frontend url from your distributor)</td>
</tr>
<tr>
<td>Number of simulation nodes</td>
<td></td>
</tr>
</tbody>
</table>
Cloud Execution

View progress in “Test Result Module” of the distributor
Cloud Execution

Cloud execution only has benefit for long running scenarios

![Graphs showing time vs number of simulation nodes for deterministic and stochastic scenarios with 100 and 20 orders.]