

Grant Agreement No.: 604656

Project acronym: NanoSim

Project title: A Multiscale Simulation-Based Design Platform for Cost-Effective CO₂ Capture Processes using Nano-Structured Materials (NanoSim)

Funding scheme: Collaborative Project

Thematic Priority: NMP

THEME: [NMP.2013.1.4-1] Development of an integrated multi-scale modelling environment for nanomaterials and systems by design

Starting date of project: 1st of January , 2014

Duration: 48 months

WP N°	Del. N°	Title	Version	Lead beneficiary	Nature	Dissemin. level	Delivery date from Annex I	Actual delivery date dd/mm/yyyy
1	7	Porto Automated Test System	0.2	SINTEF	Report	PU	12	05/06/2015

1 Executive Summary

The Porto Automated Test System (ATS) is a plugin for the Portoshell scripting environment for managing the execution of workflows. The test system can utilize all features of the Porto platform in the task of running scheduled computations - such as employing the code/text generator, available post-processing plugins (where available), managing data with the MongoDB^[1] backend and off course execute custom scripts. The workflow manager is built on the powerful State Machine Framework in Qt5^[2] which proves a hierarchical finite state machine (HSM). The execution algorithm is based on the State Chart XML (SCXML)^[3], however for harmonizing the methodology on the Porto platform we provide a way to define generic State Charts in the JavaScript Object Notation (JSON)^[4] format. The workflow implementation is multi-threaded and allows the execution of concurrent workflows with automatic utilization of all available processor cores^[5]. For reporting the results of a workflow execution the Porto.MVC module may be used for generating e.g. HTML, LaTeX with support of tools such as gnuplot, Google Chart^[6], ++.

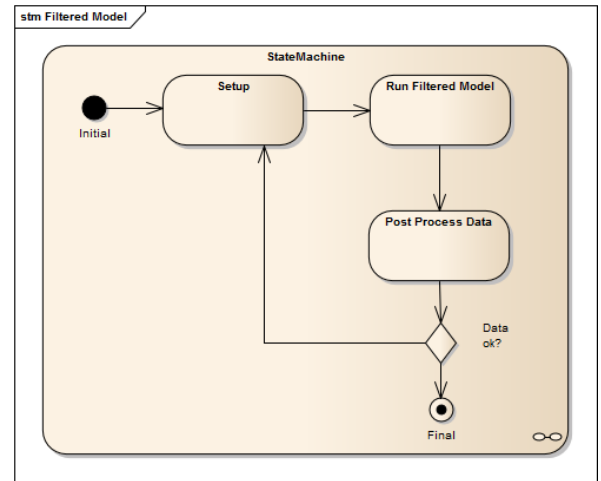


Figure 1 - A State Chart (UML) defining a workflow with a conditional transition

2 The Porto ATS

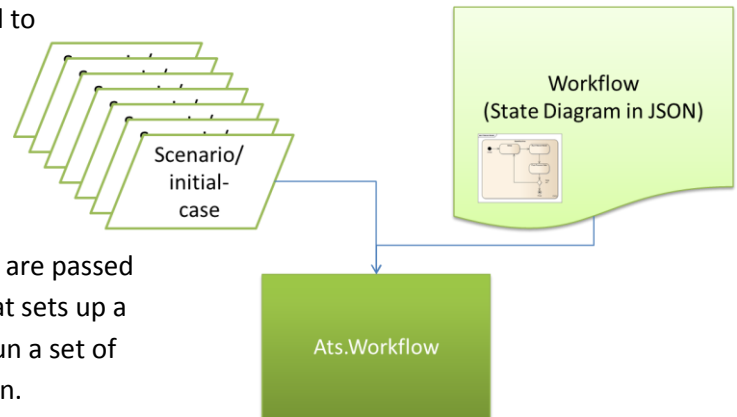
The current Porto ATS (v0.2) is based on the SOFT5 ATS (SINTEF internal, LGPL licensed) that has been employed as a testing environment for the commercial software LedaFlow^[7]. This test system is running nightly concurrent tests on hundreds of test-cases and is used for automatic generation of detailed test reports in the PDF-format and summary reports in HTML5 (Figure 2). The Porto ATS is designed to be a more generic tool that supports complex workflows with conditional transitions (branches and loops) based on a Hierarchical State Machine. This has been necessary to support the workflows in the currently defined use cases. Since the Hierarchical Finite State Machine is theoretically "Turing Complete"^[8] there are no technical limitations to the complexity of the workflows possible to define, and should therefore suffice for all possible future configurations.

2.1 Test System Requirements

The Porto ATS is provided as a generic test-runner for everything from very simple to complex workflows with conditional transitions and loops. This chapter outlines some of the key requirements of the testing environment.

2.1.1 Run a single workflow on multiple instances

One key concept of the data centric architecture is that the simulators are adapted to the data and only a unique identifier (UUID^[9]) needs to be passed to the simulation code (or wrapper) as a command line argument. The unique identifier points to an entity or a collection. When the simulator receives the UUID, it is able to read and interpret the data contents and start the simulation. When multiple simulators are connected in a workflow, only UUIDs are passed between the processes, not files. The initial data that sets up a "scenario" or "case" must exist. The ATS is able to run a set of scenarios/cases using the same workflow description.



2.1.2 Automatic scalability to the number of processes available

The ATS needs to be able to adjust the number of threads used for running concurrent workflows according to the number of processor cores available. In addition it needs to support for a manual override.

2.1.3 Fully integrated in the Porto scripting shell environment

There will often be a need to make local adjustments to a given testing environment and having the entire scripting language with all features and plugins available is a powerful feature. Extensions to the scripting environment will become immediately available for the ATS.

2.1.4 Support for regression testing, acceptance testing, performance tests and system tests.

The ATS is a tool to support the development, deployment and maintenance phases of the simulators. During development it often required to identify what impact a change will have on results, and compare this with previous versions (regression testing). Other pre-deployment tests are acceptance tests that compare results to previously defined acceptance criteria. Performance tests and system tests can be performed at any stage in the lifetime of software system.

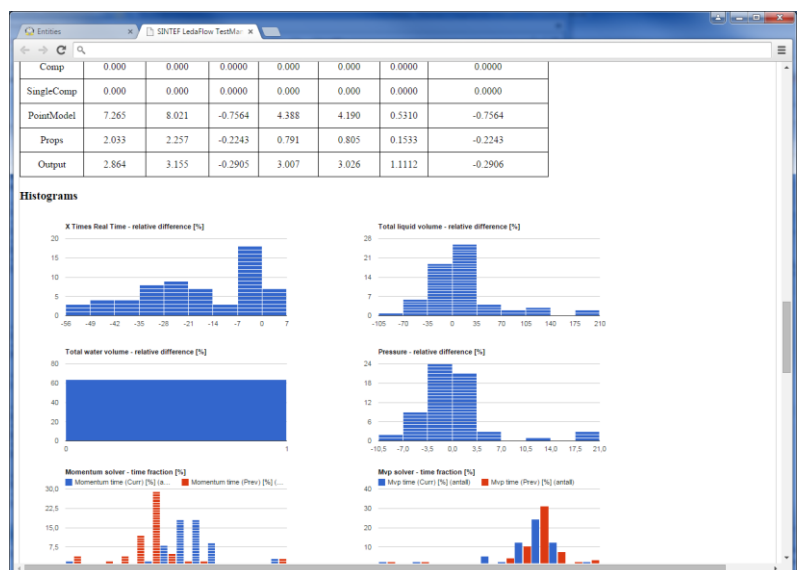


Figure 2 - Generated Summary Report

2.1.5 Explicit declaration of the State Chart

In order to effectively work and maintain the scriptable test-cases it is imperative that the work-flow definition and scenario-definitions are separated from the test runner. This allows for different workflows to be created, stored and run independently without having to maintain multiple copies of specially tailored scripts. It will also be possible (if applicable) to run the same scenarios on different workflow definitions.

See chapter [ref 2.3 State Machine Schema in JSON] for the definition of the formal schema for defining state charts.

2.2 Prerequisite

Before the Porto ATS can be employed the following prerequisites need to be met:

- Meta-Data – All necessary meta-data has been registered
- Collections – All collections of entities and interdependencies has been identified
- Storage – The data backend must be available for all processes in the workflow and the necessary driver(s) are provided.
- Workflows - The workflows must be defined (see next chapter)
- The initial case definitions (scenarios) are defined
- The ATS plugin is compiled and successfully loaded in the Portoshell environment

2.3 State Machine Chart Schema in JSON

The W3C has Proposed a Recommendation (30th april 2015) for State Chart extensible Markup Language (SCXML). A sensible choice would be to follow this recommendation. However, as we've settled on JSON instead of XML as our data-interchange format due to its many advantages, we are basing our State Charts on JSON (semantically equal to SCXML). There are no limiting factors of supporting both (or make a switch) in the future.

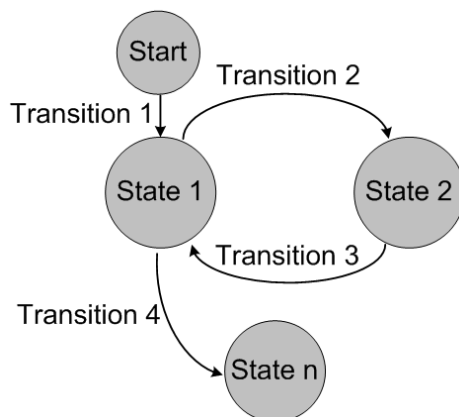


Figure 3 - States and transitions

The main idea of the State Chart is to identify how signals (triggers) cause a transition from one state to another (Figure 3). In our context the *state* represents the execution of a given simulation in our workflow. Based on the success or variable outcomes of a simulation we can define different possible transitions that will lead to the execution of a different (or the same) simulator.

In the next section we've included the formal schema for declaring state chart definitions in JSON

2.3.1 Formal JSON-schema for State Chart Definitions

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "id": "http://jsonschema.net",
  "type": "object",
  "properties": {
    "name": {
      "id": "http://jsonschema.net/name",
      "type": "string"
    },
    "version": {
      "id": "http://jsonschema.net/version",
      "type": "string"
    },
    "initial": {
      "id": "http://jsonschema.net/initial",
      "type": "string"
    },
    "senders": {
      "id": "http://jsonschema.net/senders",
      "type": "array",
      "items": {
        "id": "http://jsonschema.net/senders/0",
        "type": "string"
      }
    },
    "states": {
      "id": "http://jsonschema.net/states",
      "type": "array",
      "items": {
        "id": "http://jsonschema.net/states/0",
        "type": "object",
        "properties": {
          "id": {
            "id": "http://jsonschema.net/states/0/id",
            "type": "string"
          },
          "final": {
            "id": "http://jsonschema.net/states/0/final",
            "type": "string"
          },
          "transitions": {
            "id": "http://jsonschema.net/states/0/transitions",
            "type": "array",
            "items": {
              "id": "http://jsonschema.net/states/0/transitions/0",
              "type": "object",
              "properties": {
                "sender": {
                  "id": "http://jsonschema.net/states/0/transitions/0/sender",
                  "type": "string"
                },
                "event": {
                  "id": "http://jsonschema.net/states/0/transitions/0/event",
                  "type": "string"
                },
                "target": {
                  "id": "http://jsonschema.net/states/0/transitions/0/target",
                  "type": "string"
                }
              }
            }
          }
        }
      }
    }
  }
}
```



NanoSim

NanoSim - A Multi-scale Simulation-Based Design Platform for Cost-Effective CO2 Capture Processes using Nano-Structured Materials



```
},  
  "required": [  
    "name",  
    "version",  
    "initial",  
    "senders",  
    "states"  
  ]  
}
```

2.4 Porto ATS API

The `Ats.Workflow` provides a high-level API that simplifies the process of writing multi-threaded workflow execution applications. It is based on the Qt Concurrent module^[10] for multi-threading, the Qt State Machine Framework^[2] for the execution of the workflows and the QtScript module^[11] for embedding everything into the scripting platform.

`Ats.Workflow: public`

<code>run (workflow, testCases, callbackFn)</code>	Execute the <i>workflow</i> on the given <i>testCases</i> and Invokes the <i>callbackFn</i> .
<code>environment</code>	JS-map containing environmental variables that will be available for the processes during the execution of the workflow

Watcher is a monitoring class (Based on the `QFutureWatcher`^[12] class in Qt5) that uses signals and slot mechanisms for asynchronous status updates on the items it is watching. It allows for callback functions that gets invoked when the watcher emits signals^[13].

`Watcher: [public slots]`


<code>cancel()</code>	Cancels the asynchronous computation
<code>pause()</code>	Pauses the asynchronous computation.
<code>resume()</code>	Resumes the asynchronous computation.
<code>togglePaused()</code>	Toggles the paused state of the asynchronous computation.

`[signals]`

<code>cancelled()</code>	The watched job is cancelled.
<code>finished()</code>	The watched job finishes.
<code>paused()</code>	The watched job is paused.
<code>progressRangeChanged (min, max)</code>	The progress range changes to [min,max].
<code>progressTextChanged (progressText)</code>	The progress text has changed.
<code>progressValueChanged (progressValue)</code>	The progress value has changed.
<code>resultReadyAt (index)</code>	The job reports ready result.
<code>resultsReadyAt (beginIndex, endIndex)</code>	Reports multiple ready results.
<code>resumed()</code>	The watched job is resumed.
<code>started()</code>	The watched job has started.

2.5 Script for running automated testing

The following script is an example of a generic workflow test runner script.

```
#!/usr/bin/env portoshell
/*
-----

-----
A generic script for running concurrent test-scenario work-flows.
-----
*/

__main__ = function (args)
{
  /* Define a local event-loop as we're running asynchronously.*/
  var eventLoop = new EventLoop();

  /* Parses and reads the file containing the State Chart JSON. */
  var workflow = new StateChart(args[1]);

  /* The test cases contains data source URI, UUID and driver options
  (db name, collection name) */
  var testcases = new readTestCases(args[2]);

  /* Set environmental variables in the process execution scope. */
  Ats.Workflow.environment["ATS_PATH"] = "/var/porto/workspace/bin";

  /* Run the work flow */
  Ats.Workflow.run (workflow, testcases, function () {

    /* Define asynchronous call-back functions that are triggered
    by the watcher-object signals */

    watcher["started()"].connect (function (){
      console.log ("Started");
    });

    watcher["finished()"].connect (function (){
      console.log ("Done");
      eventLoop.quit(); /* Leave the event loop */
    });

    watcher["resultReadyAt(int)"].connect (function (index){
      console.log ("Workflow " + index + " finished");
    });
  });

  /* Start the event loop*/
  return eventLoop.exec();
}
```


2.6 Running the ATS

The Porto ATS is intended to be executed from the command line, but is equally suited for begin run from a Continuous Integration System such as Jenkins or as a web-service through the established Common Gateway Interface (CGI) protocol.

2.7 Reporting

The ATS doesn't include a reporting tool, but the ability to query databases and generate text documents (Porto.MVC) makes for an easy utilization of many available reporting tools. One possible application of this can be the generation of HTML5 Google Chart enabled Documents, that not only allows for displaying a rich variety of graphs and tables, but also includes interactive widgets for filtering the displayed data. Other options are to generate DocBook/AsciiDoc or LaTeX documentation with graphics produced with tools such as GNUPlot^[14] and Graphviz^[15].

During the workflow it recommended to transfer the relevant computational data to the "internal" database (MongoDB) for reporting. This enables the data for 1-to-1 comparison when the same scenarios are simulations are re-run with updated versions of the software. The reporting step can be included as part of the workflow, or as a standalone tool that is invoked when needed. The latter option can even be incorporated in a web-application for improved usability.

3 Summary

The Porto ATS is a generic and powerful workflow runner aimed towards developers that needs to run and re-run large batches of predefined scenarios and workflows. The workflow-runner is implemented on an established Hierarchical State Machine Framework and embedded in the Portoshell scripting utility. The current version is premature for production work, but is actively improved.

1. MongoDB Inc. *The MongoDB 3.0 Manual*. 2015; Available from: <http://docs.mongodb.org/manual/>.
2. Qt Company Ltd. *The State Machine Framework*. 2015; Available from: <http://doc.qt.io/qt-5/statemachine-api.html>.
3. W3C. *State Chart XML (SCXML): State Machine Notation for Control Abstraction*. 2015; Available from: <http://www.w3.org/TR/scxml/>.
4. D. Crockford. *RFC 4627 - The application/json Media Type for JavaScript Object Notation (JSON)*. 2006; Available from: <http://www.ietf.org/rfc/rfc4627.txt>.
5. Margaret Rouse. *Definition: multi-core processor*. 2007 [cited 2013 March 6]; Available from: http://searchdatacenter.techtarget.com/sDefinition/0,,sid80_gci1015740,00.html.
6. Google Inc. *Display live data on your site*. Available from: <https://developers.google.com/chart/>.
7. Technologies, K.O.G. *Flow Assurance*. Available from: <http://www.kongsberg.com/en/kogt/products%20and%20services/flow%20assurance/>.
8. Belzer, J.H., Albert George; Kent, Allen, *Encyclopedia of Computer Science and Technology*. Vol. 25. 1975: USA: CRC Press.
9. Society., T.I., *RFC 4122 - A Universally Unique Identifier (UUID) URN Namespace*. 2005.
10. Qt Company Ltd. *Qt Concurrent*. 2015; Available from: <http://doc.qt.io/qt-5/qtconcurrent-index.html>.
11. Qt Company Ltd. *Qt Script*. 2015; Available from: <http://doc.qt.io/qt-5/qtscript-index.html>.
12. Qt Company Ltd. *Qt Documentation*. 2015; Available from: <http://doc.qt.io/qt-5/>.
13. Qt Company Ltd. *Signals & Slots*. Available from: <http://doc.qt.io/qt-5/signalsandslots.html>.
14. GNU. *Gnuplot Homepage*. 2015 [cited 2015 June 15]; Available from: <http://www.gnuplot.info/>.
15. Arif Bilgin, et al. <http://www.graphviz.org/>. 2015 [cited 2015 June 5].