

SALADYN

Software User Guide

ANR Project Saladyn

ANR-08-COSI-014
Liverable 2c.2

Date : June 24, 2011

Table of Contents

<u>0.1 Product Release Information</u>	1
<u>1 Introduction</u>	3
<u>2 Installation</u>	3
<u>2.1 Pre-requisites</u>	3
<u>2.1.1 Code ASTER</u>	3
<u>2.1.2 LMGC90</u>	3
<u>2.1.3 Siconos</u>	3
<u>2.1.4 Graph library</u>	3
<u>2.1.5 The Salome platform</u>	3
<u>2.2 Obtaining the Saladyn toolbox</u>	4
<u>2.3 Installation of the Toolbox</u>	4
<u>2.3.1 Install from source</u>	4
<u>2.3.1.1 Install the python package</u>	4
<u>2.3.1.2 Patch ASTER</u>	4
<u>2.3.2 Test installation</u>	5
<u>3 End-users</u>	5
<u>3.1 ASTER/LMGC90 coupling</u>	5
<u>3.1.1 User data</u>	6
<u>3.1.2 Layout</u>	6
<u>3.1.3 Launch procedure</u>	6
<u>3.1.4 Post process</u>	7
<u>3.2 SICONOS/LMGC90</u>	9
<u>4 Advanced Features</u>	9
<u>4.1 Setup the graph</u>	10
<u>4.2 Typical use of the TimeSteppingSimulation class</u>	10
<u>4.2.1 Typical usage no 1:</u>	10
<u>4.2.2 Typical usage no 2:</u>	10
<u>4.3 Detailed Documentation</u>	title

Saladyn User Guide

1 Introduction

The Saladyn project (ANR-08-COSI-014) aims at providing a framework for coupled computations in the field of deformable multi-body systems with non-smooth interactions.

This is the user guide for the python Saladyn Toolbox. Since developpement is in progress, we describe here the current setup and procedure used to launch coupling simulations.

This document is labeled 2c.2 in the ANR-08-COSI-014 formalism.

Please refer to document 2c.1 for the detailed description of classes and methods.

Saladyn User Guide

2 Installation

Installation has been tested on the following systems:

- Linux Debian Squeeze

2.1 Pre-requisites

The prerequisite is to have a working installation of the softwares to be coupled:

- Code_ASTER (EDF R&D)
- LMGC90 (LMGC - Université de Montpellier)
- Siconos (INRIA Bipop)
- a python graph library (see below).

One or two of these softwares can be skipped, depending on the type of computations needed.

The user is referred to the corresponding software documentation for proper installation.

2.1.1 Code_ASTER

Code_ASTER needs to be installed. The Saladyn toolbox is tested with versions 10.0.x and 10.2.x. The ASTER tools must be on the exec path (specifically as_run).

2.1.2 LMGC90

LMGC90 needs to be installed, and python libraries must be on the python path. The python interface library named chipy must be on the python search path. The Saladyn platform has been tested with LMGC90 revision no 2393.

2.1.3 Siconos

Siconos needs to be installed, and on the python path.
Version(s) ?

2.1.4 Graph library

A python graph library must be installed. The Saladyn platform has been tested with two of the available free graph libraries:

- the BGL python binding for graph-tool;
- the pygraph library.

2.1.5 The Salome platform

The Salomé platform is not mandatory, but can be used as a post and pre processing tool.

2.2 Obtaining the Saladyn toolbox

Currently the only way to obtain the toolbox is by developer svn access to the repository at INRIA.

Future release may include:

- binary packages
- source packages
- Virtual machine with complete pre-install of LMGC90, ASTER, Siconos and Saladyn

2.3 Installation of the Toolbox

Since the only distribution medium is currently in the form of svn source code, we describe only the installation from source.

2.3.1 Install from source

There are two major steps: install the python packages in a sytem wide location and patch the ASTER installation.

2.3.1.1 Install the python package

We use python distutils for the installation of the toolbox as a python package. In the Toolbox top directory, do as root:

```
.../Toolbox# python setup.py install
```

Default install location is in `/usr/lib/python2.x/dist-packages` according to the version of python found.

This can be changed with the usual `--prefix` option to the `setup.py` script.

2.3.1.2 Patch ASTER

In the Toolbox top directory, issue:

```
.../Toolbox$ python setup.py patch
```

This installs a user local patched version of ASTER into the local directory `Toolbox/TestsAster/Patch`.

As a side effect, this script also prepares the ASTER run scripts inside the test directories.

Note: if you gave the `--prefix` option to the install procedure, you should give the same `--prefix` option to this command.

TODO PATCH --version=NEW10

2.3.2 Test installation

The Saladyn toolbox comes with a series of basic tests. In order to run them, you should issue the following command:

```
.../Toolbox$ python setup.py test
```

3 End-users

This section is end-user oriented. We give the global outline for each type of coupled simulation. The user is referred to the examples for details.

Some examples of ASTER/LMGC90 coupling are given:

- in the `TestsAster` directories (`T003`, `T004`, `T006`),
- also in the benchmarks use cases "Rocking", "ThreeCubes", "Marbles"

One example of LMGC/SICONOS coupling is given in the `TestsLmgcSiconos` directory.

Examples of Siconos usage in the `TestsIntegrationSiconos` directory.

3.1 ASTER/LMGC90 coupling

For these couplings, we need to setup data for both LMGC90 and `Code_ASTER`.

Once data is setup as described in the next section, the launch procedure consists in launching `Code_ASTER`, with initial ASTER input script `AsterLmgc90Script.py` located in the `Toolbox/data` directory. The user only furnishes The ASTER commands that are left for the user are those that define the ASTER meshes, materials, loads, and boundary conditions.

3.1.1 User data

All needed data are grouped in one directory, e.g. `data/`. We need

- LMGC90 data, which comes in a directory `data/DATBOX/`, containing LMGC90 input file `BODIES.DAT`, `TACT_BEHAV.DAT`, `MODELS.DAT`, etc. In the ASTER/LMGC90 coupling scheme, we use LMGC90 not only as a rigid body modeller, but also as a contact detector. As a consequence, these files must define all physical bodies and tactors, even meshed bodies modelled by ASTER.
- The output directories named `DISPLAY`, `POSTPRO`, and `OUTBOX` must also be created besides the `DATBOX` directory (use `mkdir-lmgc`).
- `Code_ASTER` user data, which comes in the form of a python module in the `data/` directory, called `AsterUserData.py`. This user defined python module must
 - ◆ be called `AsterUserData.py`,
 - ◆ use aster commands to define the needed aster data structures, with imposed names: `asModel`, `asMesh`, `asMater`, `asKinco`, `asLoad`,
 - ◆ load Saladyn nodes in the graph, by defining a callback `loadPhysicalObjectFromAster(graph)` whose role is to fill the graph with ASTER modelled physical objects.
- Saladyn simulation data, which is a simple python module named `SaladynUserData.py` in the `data/` directory, and which must define python variables named `t0`, `tf`, `deltaT`, respectively, initial instant, final instant, time step increment.

In this type of coupling, interactions are not brought into the Saladyn graph structure. For performance reason, LMGC90 does all the job of managing the interactions, and solving the non-smooth problem.

Note that meshes, initial conditions, boundary conditions, must be coherent between the LMGC90 input `DATBOX` abd the ASTER input python module.

Also, in order to write the `loadPhysicalObjectFromAster` callback, one must know the meshed body indices in LMGC90. This limitation will be removed in the future.

3.1.2 Layout

The typical layout of user data, as used in the test cases and examples, is as follows:

```
data/
-->DATBOX/
---->BODIES.DAT
      BULK_BEHAV.DAT
      DOF.INI
      DRV_DOF.DAT
      GPV.INI
      MODELS.DAT
      POSTPRO.DAT
      TACT_BEHAV.DAT
      Vloc_Rloc.INI
-->DISPLAY/
-->POSTPRO/
-->OUTBOX/
-->AsterUserData.py
-->SaladynUserData.py
-->mesh files (.med, .gmsh, ... ) for Code_ASTER
```

NOTE: LMGC90 data in `data/DATBOX/BODIES.DAT` and `Code_ASTER` data must be coherent (identical meshes). No check is made.

3.1.3 Launch procedure

The launch procedure is the same as for launching `Code_ASTER`: either through the `astk` interface, or writing the `.export` file by hand.

For both method, the following should be observed:

- The input script to use for this `Code_ASTER` run is `..../Toolbox/data/AsterLmgc90Script.py`.
- One must add as input directory for `Code_ASTER` the `./data` directory in the above layout.
- One must add an output directory, in which results will be found. It is recommended that it be different from the `./data` directory.
- One must use the patched version of `ASTER` executable and `ASTER` catalogs in directory `..../Toolbox/TestsAster/Patch`.
- the `ASTER` results files (`.mess`, `.resu`, `.med`, etc...) can be placed anywhere.

Refer to existing `.export` or `.export.template` file in examples for more details.

If using `astk`: click "run".

If using hand written `.export` file run the command `"as_run myCase.export"`.

3.1.4 Post process

Results of the simulation are layed out as follows:

- the given output directory contains the results for LMGC90: `DATBOX`, `POSTPRO`, `OUTBOX`, `DISPLAY` directories, filled with the results of the simulation, as given in the `POSTPRO.DAT` file.
- the `ASTER` result files (`.mess`, `.resu`, etc...) are as given in the `ASTER` run setup.

3.2 SICONOS/LMGC90

4 Advanced Features

This section is more developer oriented.

The Saladyn toolbox can be used as a python library. Python scripts can be entirely user defined, using the following outline:

- Define the graph, with nodes, mechanical models, interaction laws and edges (interactions);
- define the modelers, attach to the graph;
- define the time discretization and the simulation object, attach them to the graph;
- define the contact detector object, attach it to the graph;
- Pilot the simulation from the script.

Refer to the examples for details.

4.1 Setup the graph

The steps to follow are:

- create the graph;

```
from Saladyn import Graph
myGraph=Graph()
```

- fill it with bodies and mechanical models;

```
po = PhysicalObject.PhysicalObject(name='block',ndim=<2 or 3>)
# attach a mechanical model to the body
mo = LMGC_MechanicalModel.LMGC_MechanicalModel(name=int(id),body_type='RBDY2')
po.addMechanicalModel(mo)
# attach the dof list to the mechanical model
# rigid body has 3 degrees (2D) of freedom for 2 fields (displ,velocity)
#           or 6 degrees (3D) of freedom for 2 fields (displ,velocity)
nbdofs=3
dof=DegreeOfFreedom.DegreeOfFreedom(nbdofs,2)
mo.setDegreesOfFreedom(dof)
# Finally, add the object as a node of the graph
myGraph.addNode(po)
```

- add the interactions and interaction laws (which correspond to the index set 0 of potential interactions);
- create the time discretization and the simulation, and attach the simulation object to the graph,

```
from Saladyn import TimeDiscretization
myTimeDiscr=TimeDiscretization(tInit=0.0,nTimeSteps=1000,deltaT=0.001)

from Saladyn import SICONOS_TimeSteppingSimulation
mySimulation=SICONOS_TimeSteppingSimulation(myTimeDiscr)
myGraph.addSimulation(mySimulation)
```

- create the contact detector (geometry toolbox), contact solver, and attach them to the graph;

```
# LMGC as collision detector
from Saladyn import LMGC_InteractionBuilder
myDetector=Saladyn.LMGC_InteractionBuilder(timeDiscretization=myTimeDiscr)
myGraph.addContactDetector(myDetector)

# LMGC as contact solver
```

Saladyn User Guide

```
from Saladyn import LMGC_InteractionSolver
myContactSolver=Saladyn.LMGC_InteractionSolver(myTimeDiscr)
myGraph.addContactSolver(myContactSolver)
```

- initialize the graph

```
myGraph.initialize()
```

4.2 Typical use of the TimeSteppingSimulation class

The simulation can then be used to write custom iterations.

4.2.1 Typical usage no 1:

```
while (time loop):
    simulation.startStep(graph)
    simulation.computeOneStep(graph)
    simulation.completeStep(graph)
```

4.2.2 Typical usage no 2:

```
while (time loop):
    simulation.startStep(graph)
    simulation._timeDiscretization.incrementStep()
    simulation.initNewtonLoop(graph)
    while (newton loop):
        simulation.startNewtonIteration(graph) ## function call on modelers
        nonconvergencecriteria=simulation.NewtonComputeOneIteration(graph) ## eq. to 3 functions in Sic
        simulation.completeNewtonIteration(graph) ## function call on modelers
    simulation.completeStep(graph)
```

4.3 Detailed Documentation

Doxygen generated doc is available [here](#).