

10th SPINE Meeting: 6-8 December 2005

SPIS-UI

Integrated Modelling Environment (EMI) for Space Modelling

Final presentation

Principle and concepts

J.Forest⁽¹⁾, J.-F.Roussel⁽²⁾, A.Hilgers⁽³⁾, B.Thiébaud⁽³⁾, S.Jourdain⁽¹⁾,
M.Biais⁽¹⁾

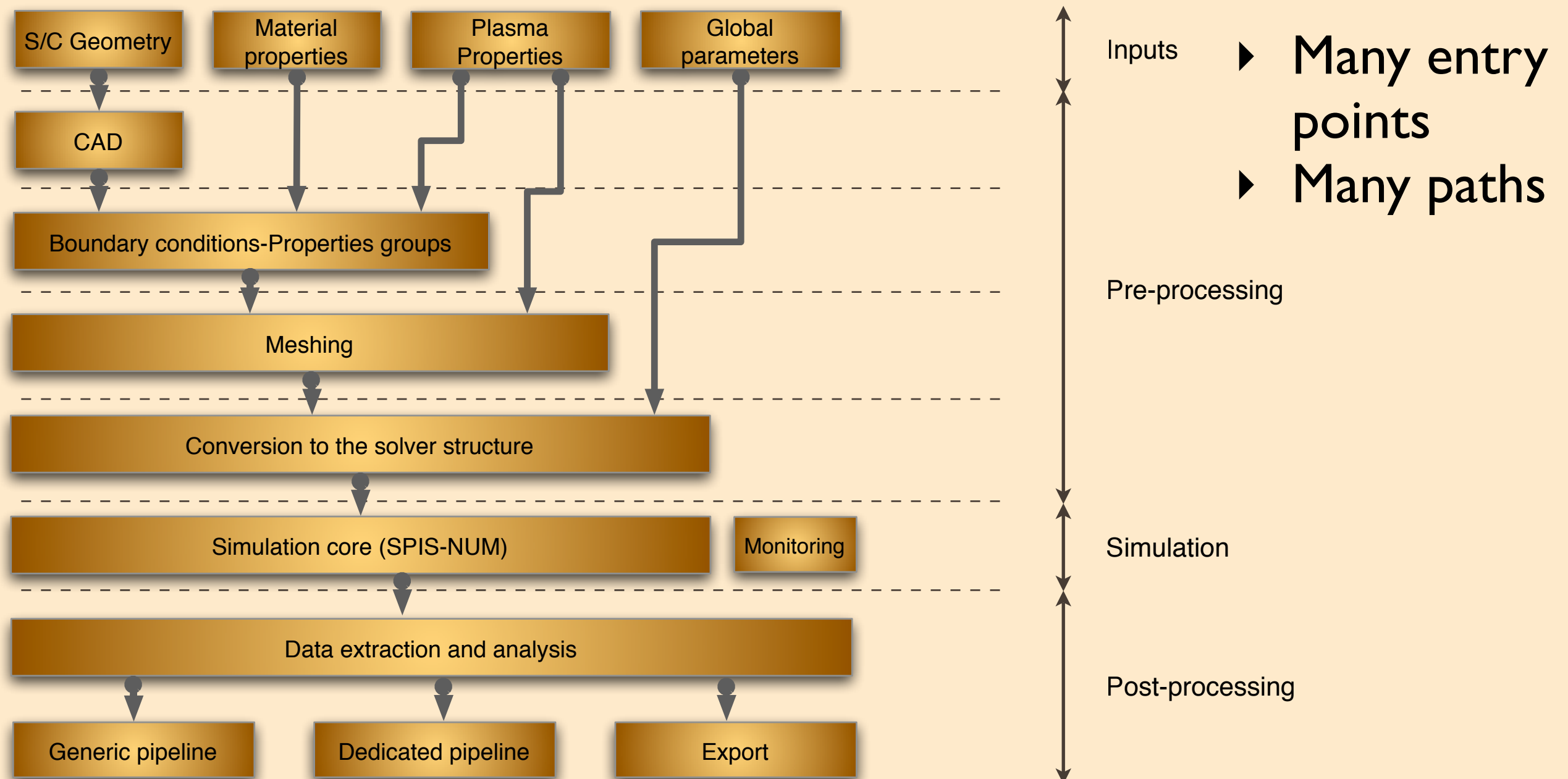
contact: contact@artenum.com

(1) Artenum, France; (2) ONERA, France; (3) ESA/ESTEC, The Netherlands

Why an advanced User Interface ?

- ▶ To be used in both scientific and engineering contexts
- ▶ To be able to perform the complete modelling chain
- ▶ Able to integrate heterogeneous tools and data in an homogenous manner
- ▶ Must be the most easy to use possible and still powerful and extensible for the advanced user
 - ◎ Many levels of control: GUI, script, source code
 - ◎ Must offer advanced pre-integrated tools
- ▶ Must be easy to use but also adaptable to various specific cases or evolutions
- ▶ Offer advanced post-processing tools for data analysis
- ▶ A base (GUI, data, actions) easy to customise
- ▶ To be able in interact with other tools (CAD, properties, env...)

Modelling chain



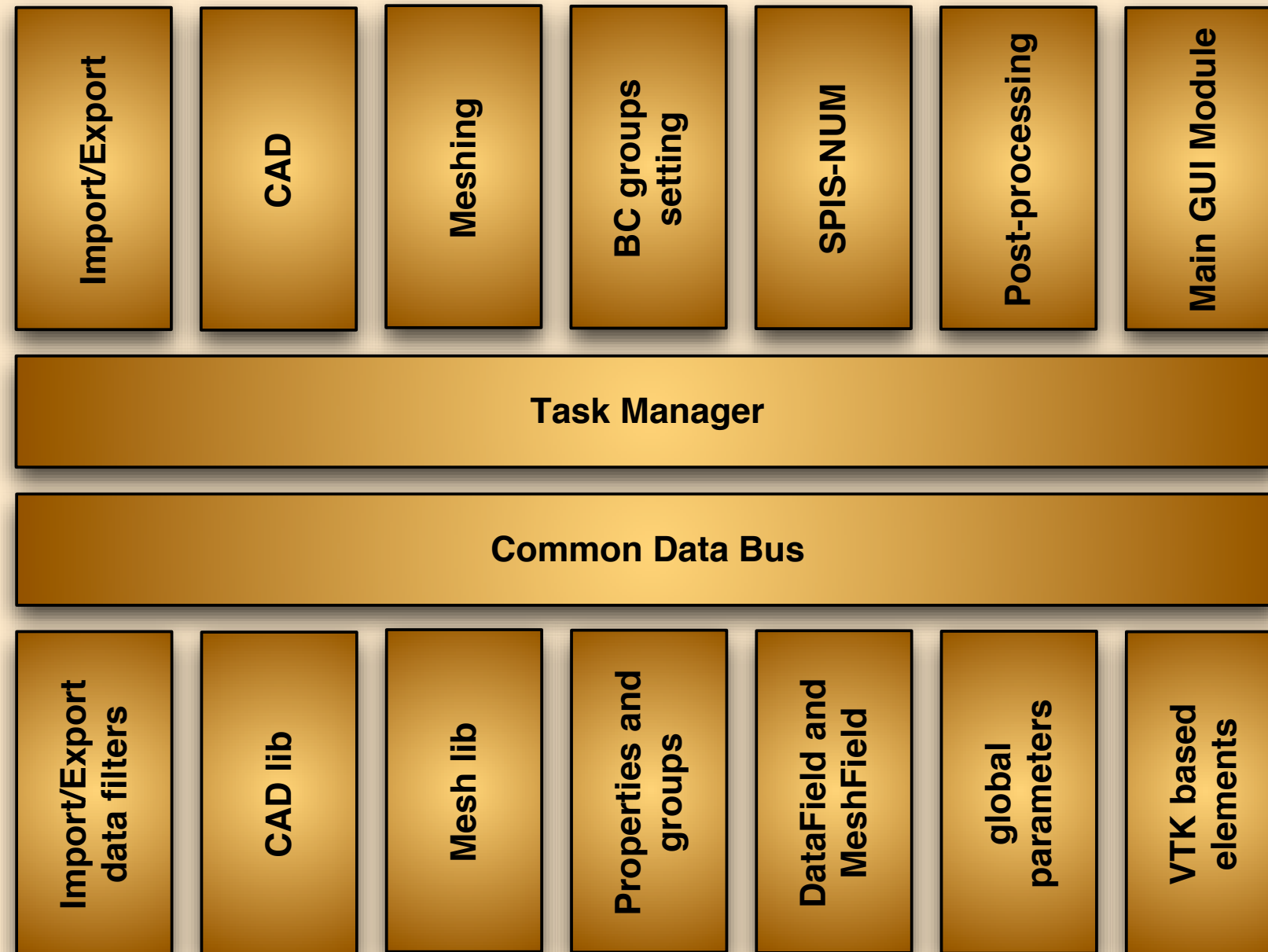
SPIS-UI as CAD/CEA framework

- ▶ An Open platform to make the link between existing and autonomous tools
- ▶ “Easy” integration for heterogeneous tools and data
- ▶ A *Common Data Bus*, to exchange and convert data
- ▶ A control module, *Task Manager*, to help the user to follow the order of the modelling process
- ▶ Many entry levels for the user:
 - ◎ GUI
 - ◎ Through a script language “à la MatLab”
 - ◎ Directly at the source code level
 - ◎ A batch mode
- ▶ Possibility of numerical models “hot-building” and data edition

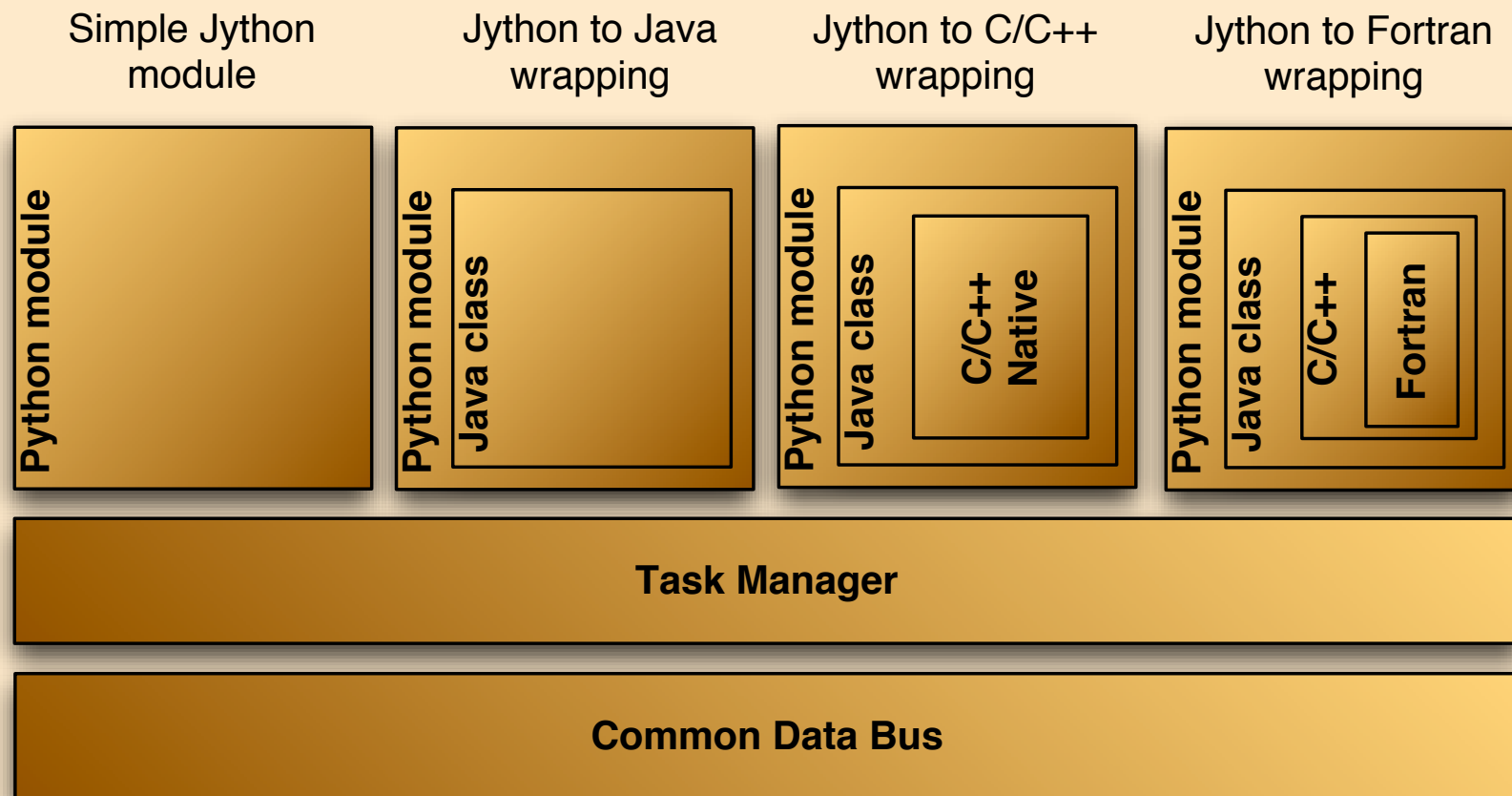
SPIS-UI Design (I)

- ▶ Each action embedded into generic *Task*
- ▶ Software components are considered as modular “plug-ins”
- ▶ A common framework to offer the GUI background, make the link between Tasks and exchange data
- ▶ Fully Java and Jython based kernel (multi-platform)
- ▶ Built-in libraries and tools:
 - ◎ CAD lib and importers/exporters
 - ◎ Mesh lib
 - ◎ Properties, BCs and controls
 - ◎ Advanced 2D/3D post-processing and visualisation
- ▶ Use external tools:
 - ◎ CAD tools
 - ◎ Tetraedric mesher (2D/3D)
 - ◎ Viewers, editors...

SPIS-UI Design (2)



How to link them together ?



Python/Jython script languages as “glue languages” to link heterogeneous elements:

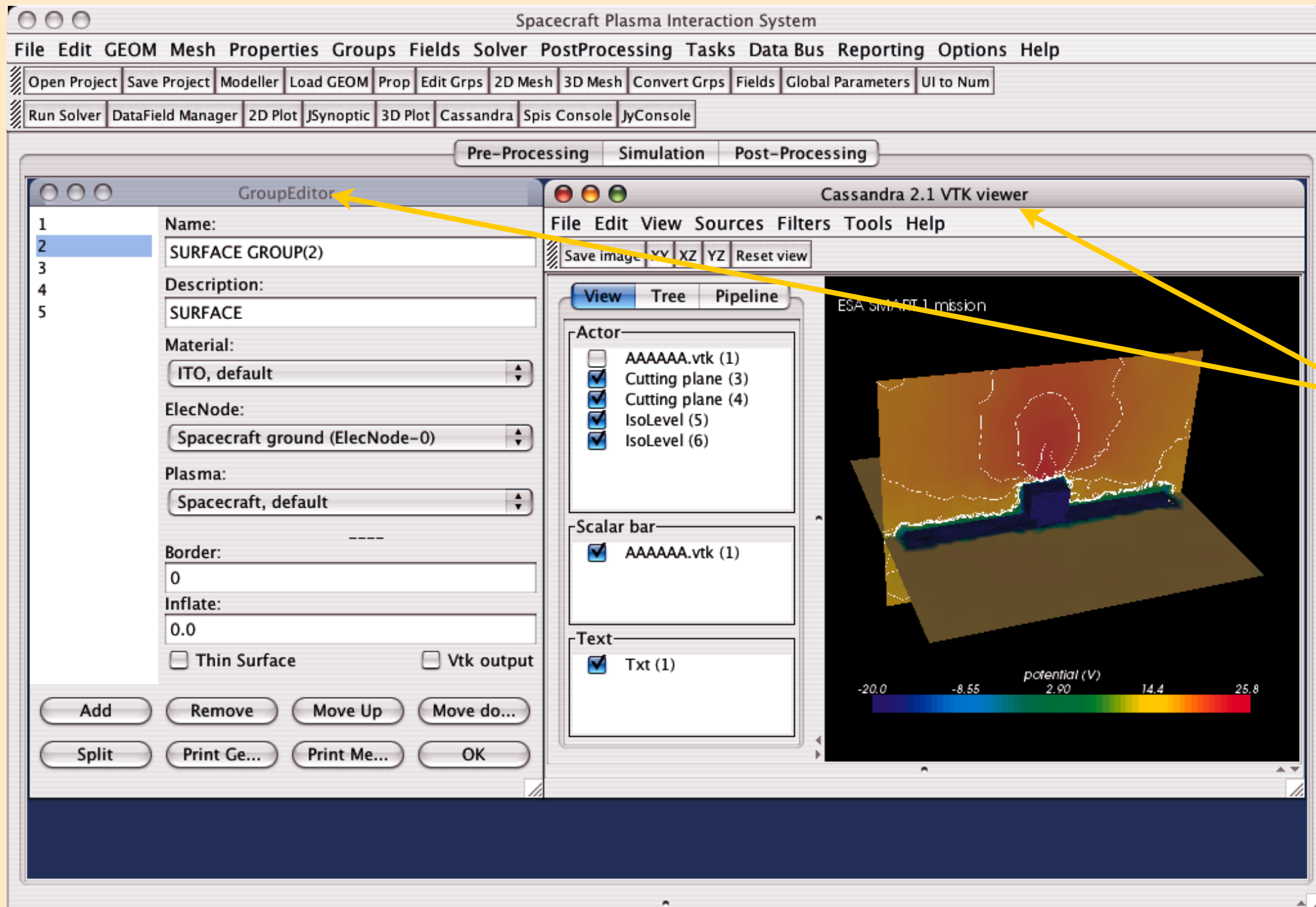
- Java
- C/C++
- Fortran

- ▶ Possibility to integrated heterogeneous elements
- ▶ Easier handling of low level codes and data
- ▶ Simpler than Corba based techniques
- ▶ Hot-plug

Technological choices

- ▶ Integration of pre-existing components (software, libraries) most as possible
 - ◎ To reduce the maintenance effort for a small community as SPINE
 - ◎ To make benefites of the dynamics of the OSS
 - ◎ To converge toward the emerging “open-standards”
- ▶ Components:
 - ◎ Jython (Script language)
 - ◎ JFreeChart and JSynoptic (2D post-processing)
 - ◎ Gmsh (CAD and meshing)
 - ◎ VTK (3D visualisation)
 - ◎ JNumerics (Jython based scientific library)
 - ◎

SPIS-UI main user interface



Menu and
tools bar

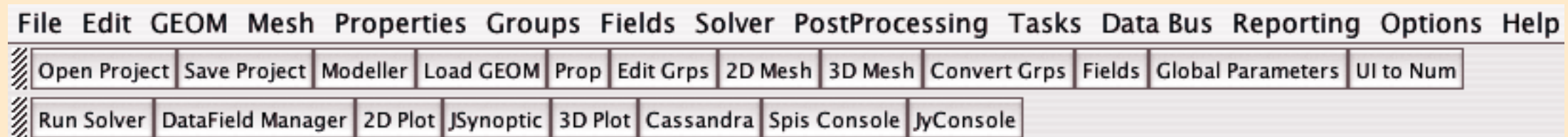
Modelling
desktops

Tasks
with their own
GUIs

Advanced
Jython console

log windows

Modelling process and *TaskManager*

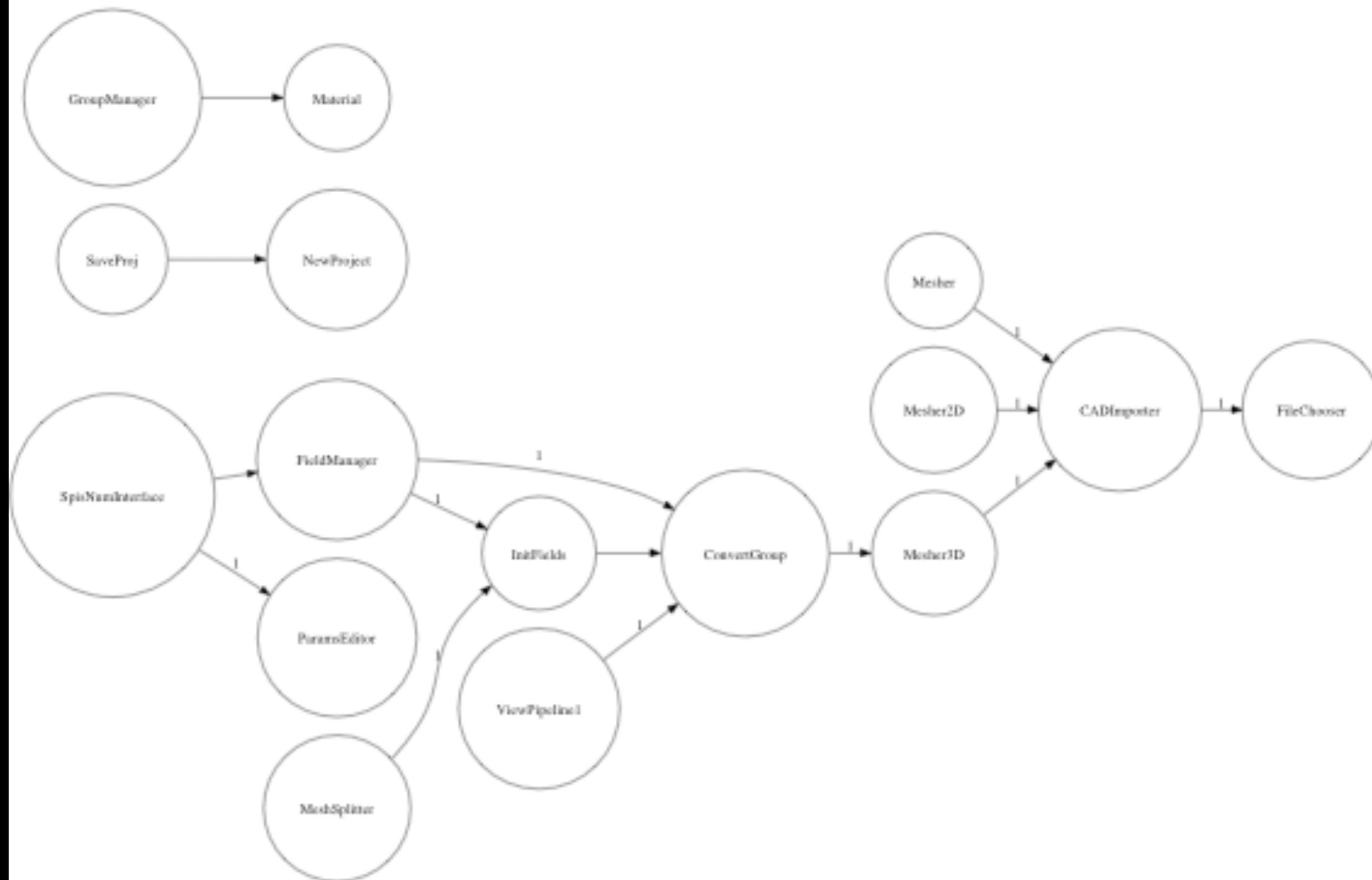


- ▶ Which button should I push ?
 - ◎ If I don't the last one.
 - ◎ If I am an advanced gamer, all of them!

The Task Manager is here for:

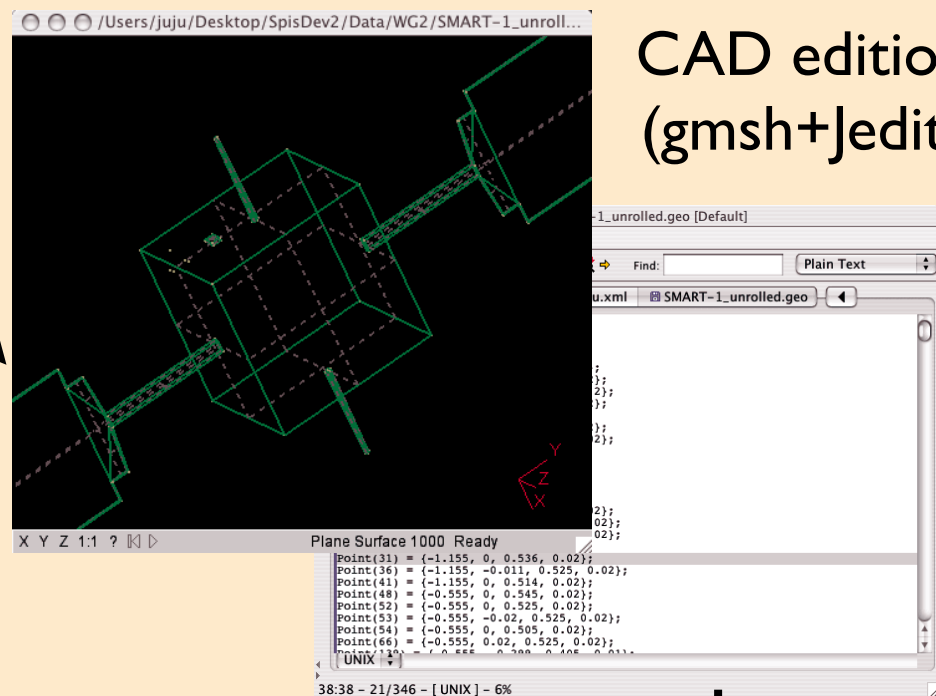
- Help the user to respect the “right order” in the performed task
- Help to maintain the data consistency
- Performs automatically the tasks needed before
- Gives awareness on already done tasks

TaskManager and dependence tree



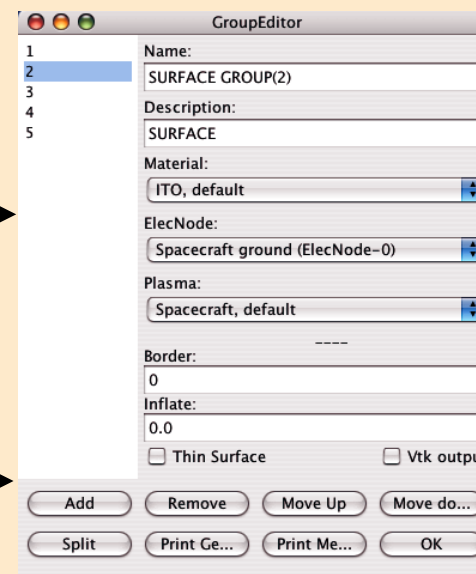
Modelling process: pre-processing

import



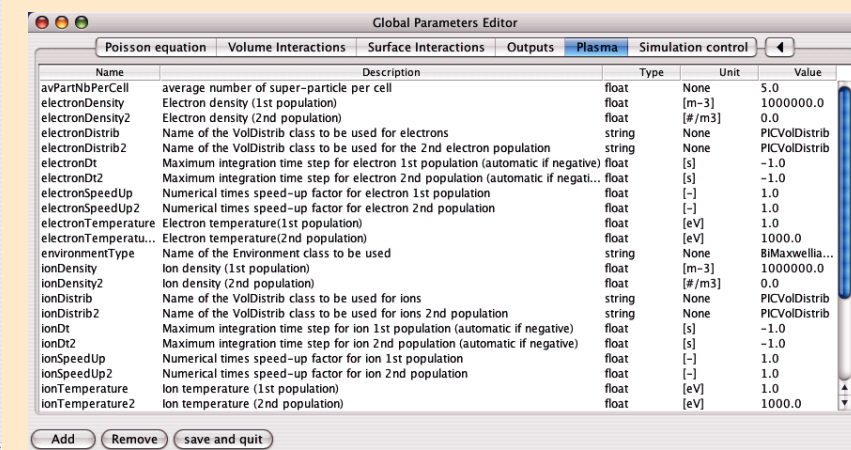
CAD edition
(gmsh+Jedit)

local parameters
and BC



SPIS-UI pre-processing modules

Global parameters



Properties

Meshing

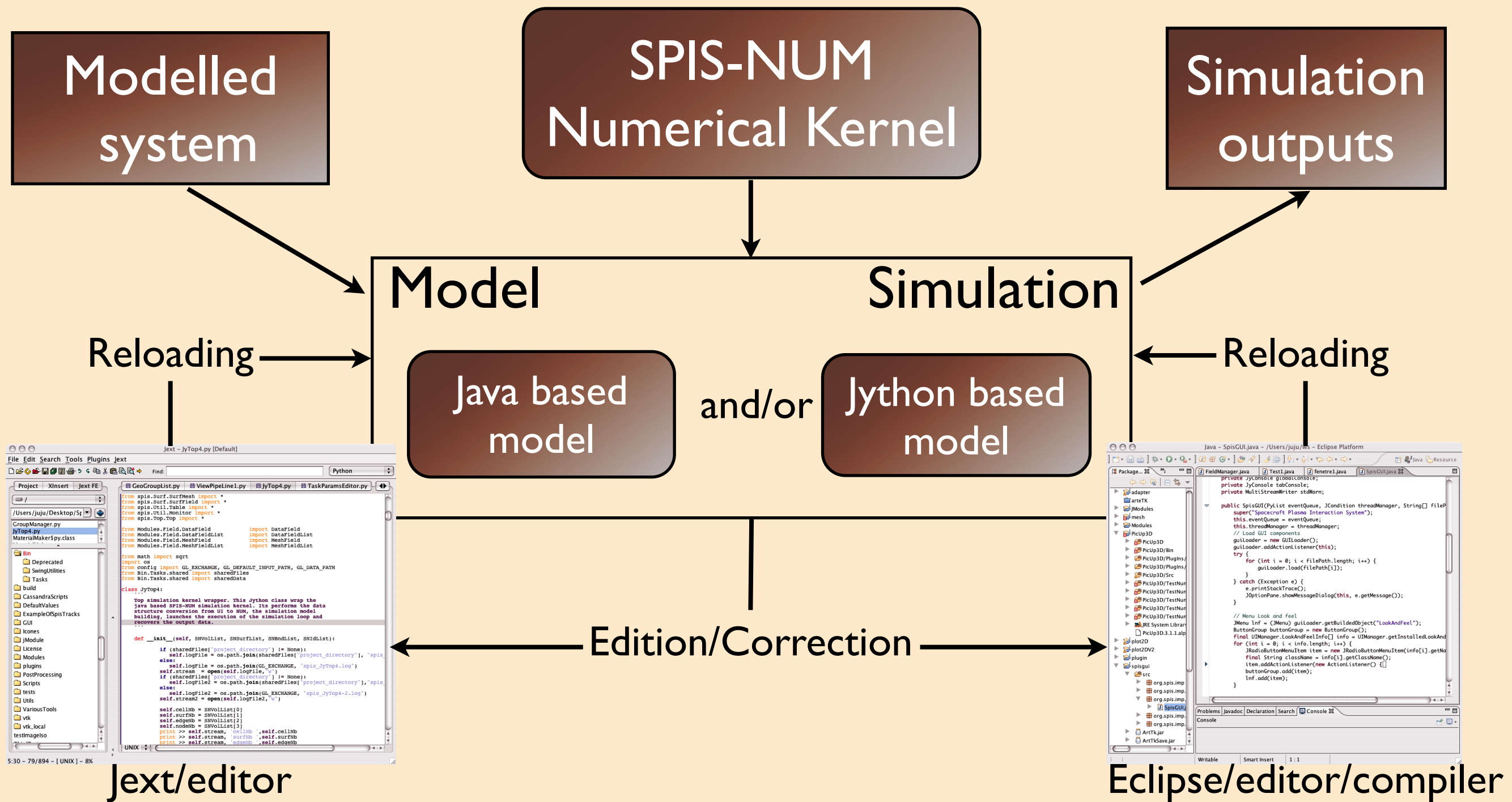
Gmsh

Netgen

Other...

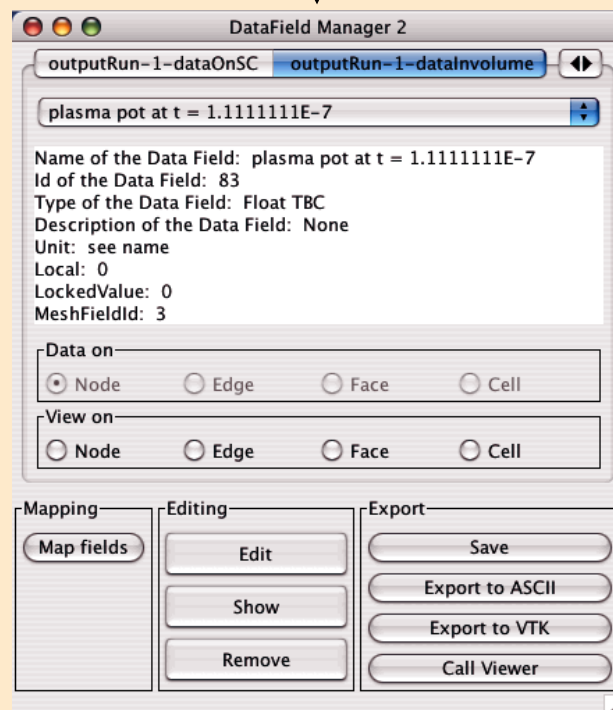
Modelled
system

Modelling process: Simulation



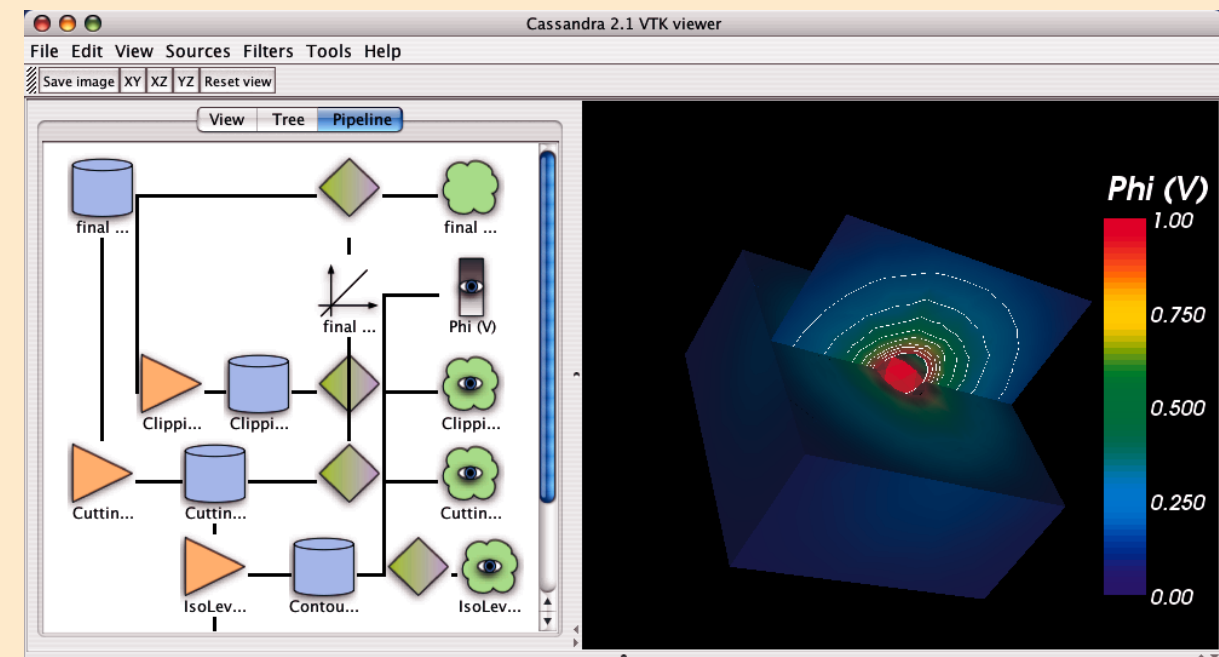
Modelling process: outputs and post-processing

Simulation
outputs

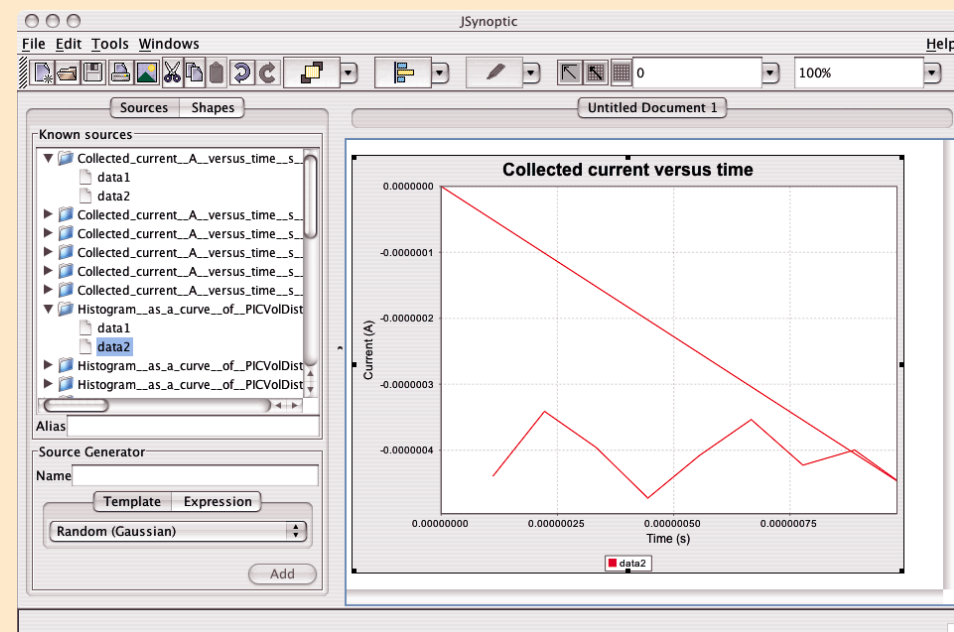


Data analysis
and conversion

3D data analysis
Cassandra
(Artenum product)
Paraview
others...



2D data analysis
SpisPlot/Jysinoptic
gnuplot, xmgrace
others...



Export
VTK/ASCII

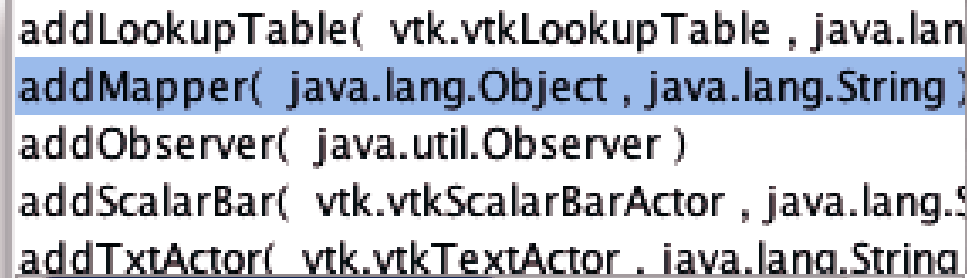
SPIS-UI as Integrated Modelling environment

- ▶ SPIS-UI is here considered as a development tool
- ▶ Science and modelling oriented
- ▶ Open design: Possibility to integrated other tools and simulation cores (e.g PicUp3D)
- ▶ Oriented toward the interaction with users, specialists and scientists
- ▶ Simplifies the design of models (i.e advanced handling of SPIS-
NUM)
- ▶ SPIS-UI must considered as an Integrated Modelling
Environment (IME) for physics
- ▶ Extensibility and application fields larger than industrial
solutions (not limited by the GUI)

JyConsole, an advanced Jython console

Welcome on JyConsole by Artenum (www.artenum.com)

```
>>> for i in range(360):  
    cassandra.getPipeLineManager().
```

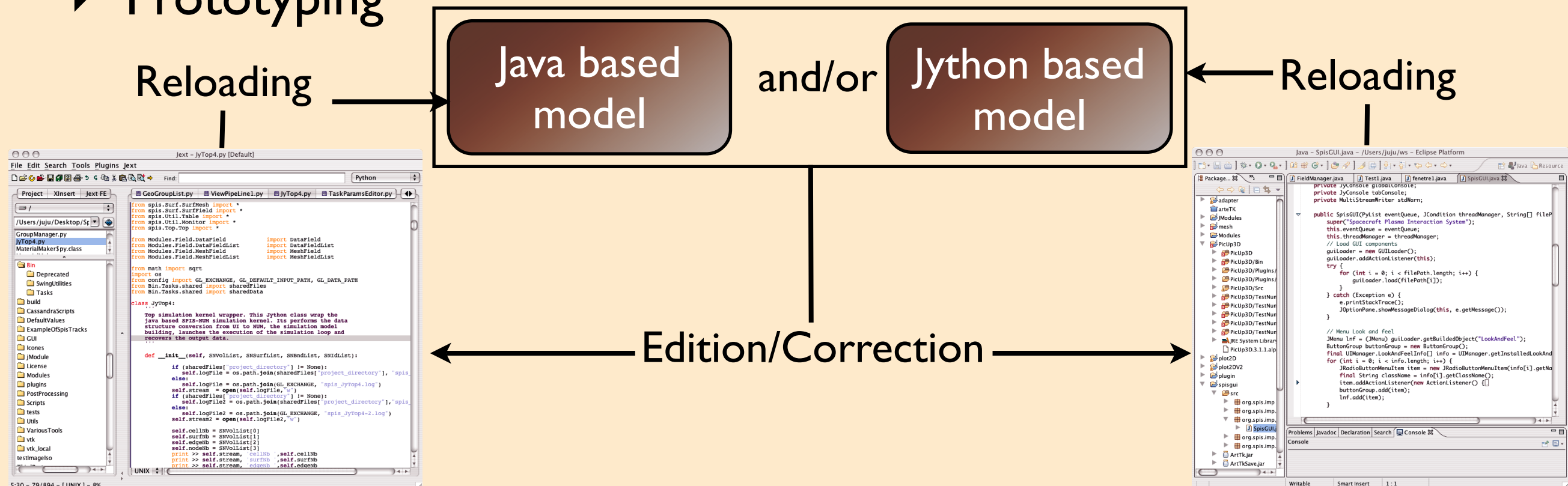


```
addLookupTable( vtk.vtkLookupTable , java.lang.  
addMapper( java.lang.Object , java.lang.String )  
addObserver( java.util.Observer )  
addScalarBar( vtk.vtkScalarBarActor , java.lang.S  
addTxtActor( vtk.vtkTextActor , java.lang.String
```

- ▶ Object oriented completion
- ▶ Data handling easier
- ▶ Low level object libraries easier to manipulate (e.g VTK)

Hot-reloading capability

- ▶ Each task can be edited, modified, reloaded during the run
- ▶ For both:
 - ◎ Jython modules (edition + reloading)
 - ◎ Java modules, e.g SPIS-NUM (edition + re-compilation + reloading)
- ▶ Allows the modification to the model
- ▶ Prototyping



Project Saving and Python serialisation

- ▶ Most of the data saved under the form of Jython modules (serialisation)
- ▶ reloading as “simple Python modules”
- ▶ Built data (pre-processing) can be used as spare-element for:
 - ◎ “other codes”
 - ◎ simples scripts

```
# Initialize Task Manager (if you want to use it))
task_manager = TaskManager(*[i[0] for i in Bin.Tasks.taskslist.tasks])
task_manager.reset_done_nodes()

# To load the projet (just by calling the LoadProj Task)
# The project should contain all needed Data. This can
# also be done by direct Python import.
project_path = "/Users/juju/AARefProject"
sharedTasks["context"] = []
sharedTasks["context"].append(project_path)
tasks_list = ["LoadProj", "FieldManager"]

# Run tasks declared in the tasklist
print "List of the tasks to be executed: %s" % tasks_list
for i in tasks_list:
    task_manager.run_tasks(i)

# now you can work on your data. They are loaded into the
# memory. For example, if you want to see the constain of the
# geo group 2
print sharedGroups['GeoGroupList'].List[2]

# just to say to the gui base task to be in batch mode
sharedTasks["context"]="batch"

# now we can continue the pre-processing stuff.
tasks_list = ["SpisNumInterface"]
print "List of the tasks to be executed: %s" % tasks_list
for i in tasks_list:
    task_manager.run_tasks(i)

#####
#      Now everything is ready to call SpisNum and perform a
#      simulation.
#####

tasks_list = ["JyTop"]
print "List of the tasks to be executed: %s" % tasks_list
for i in tasks_list:
    task_manager.run_tasks(i)

print "The Job is done ! Bye !"
```


SPIS-UI to develop itself: Integration of a new Task

Generic embedding of tasks

```
from org.spis.imp.ui.util import DirectoryDialog
from shared import sharedData

import spis

class TaskSaveProj(Task):
    """Example of task"""

    desc = "Task for the presentation"

    def run_task(self):
        """
        Example of Task
        """
        print "here we do the action"
        exemple = spis.Top.Simulation.GeoExample()
```

Keyword of control

Declaration in the dependence tree

```
# dependency tree between tasks.
tasks = [
    (TaskFileChooser("FileChooser"), 0, "FileChooser"),
    (TaskCADImporter("CADImporter", 0, "FileChooser"), 0, "CADImporter"),
    (TaskMesher("Mesher", 0, "CADImporter"), 0, "Mesher"),
    (TaskViewPipeline1("ViewPipeline1", 0, "ConvertGroup"), 0, "ViewPipeline1"),
    (TaskMaterial("Material"), 0, "Material"),
    (TaskGroupManager("GroupManager", 0, "Material"), 0, "GroupManager"),
    (TaskFieldManager("FieldManager", 0, "ConvertGroup", \
        "InitFields"), 0, "FieldManager"),
    (TaskSpisNumInterface("SpisNumInterface", 0, \
        "FieldManager", \
        "ParamsEditor"), 0, "SpisNumInterface"),
    (TaskViewPipeline2("ViewPipeline2", 1), 1, "ViewPipeline2"),
    (TaskViewPipeline3("ViewPipeline3", 1), 1, "ViewPipeline3"),
    (TaskJyTop("JyTop", 1), 1, "JyTop"),
    (TaskToolCaller("ToolCaller"), 1, "ToolCaller"),
    (TaskEditIni("EditIni", 1), 1, "EditIni"),
    (TaskPrompt("Prompt", 1), 1, "Prompt"),
```

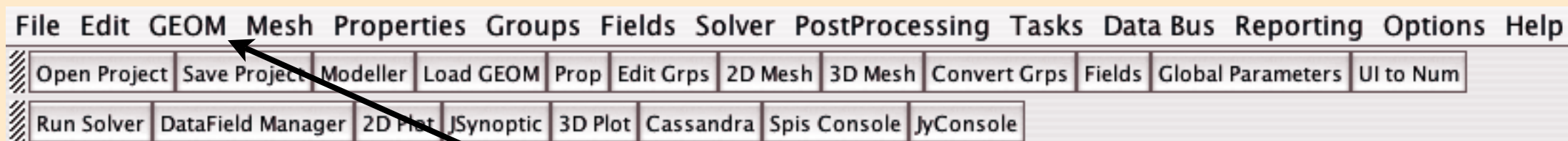
background/foreground modes

Multi-threaded kernel

SPIS-UI to develop itself: Customisation of the GUI

► XML based definition of the main GUI

- ⦿ Do not need to develop in Java
- ⦿ Do not need to be re-compiled
- ⦿ Automatically reloaded at the starting
- ⦿ Easy to modify



Keyword of control

```
<!-- ***** -->
<!-- *** GEOM *** -->
<!-- ***** -->

<GEOM className="javax.swing.JMenu" txtContent="GEOM">
  <Import className="javax.swing.JMenu" txtContent="Import">
    <geo className="javax.swing.JMenuItem" actionCommand="CADImporter" txtContent="Gmsh Geo file"/>
  </Import>

  <Export className="javax.swing.JMenu" txtContent="Export">
  </Export>

  <Modeller className="javax.swing.JMenu" txtContent="Modeller">
    <Gmsh className="javax.swing.JMenuItem" actionCommand="ToolCaller" txtContent="Gmsh"/>
  </Modeller>

  <Show className="javax.swing.JMenuItem" actionCommand="ViewPipeline1" txtContent="Show GEO groups"/>
  <Separation/>
  <Open className="javax.swing.JMenuItem" actionCommand="LoadCAD" txtContent="Open GEOM (proj)"/>
  <Save className="javax.swing.JMenuItem" actionCommand="SaveCAD" txtContent="Save GEOM"/>
</GEOM>
```

Conclusion on design and concepts

- ▶ SPIS-UI has reached enough maturity to be used by most people (It works!)
- ▶ SPIS-UI must be seen as “the missing link” between *MatLab* & *SALOME*
- ▶ Generic and multi-purpose framework
 - ◎ Adaptable to other models and software (integration of PicUp3D under development)
 - ◎ Other fields of application possible
 - ◎ Multi-physics and multi-models extension possible
- ▶ Easy to install (CD-live or), multi-platform
- ▶ Easy to adapt and extend for tailored applications
- ▶ Freely available, open source
- ▶ Based on stable technologies with a strong dynamics of community