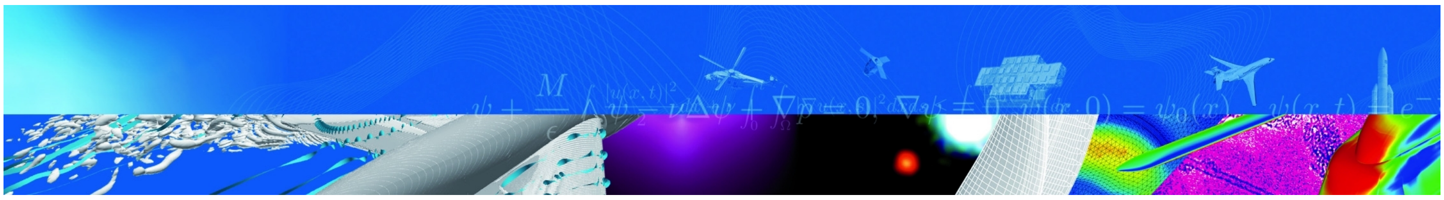


ONERA

THE FRENCH AEROSPACE LAB

return on innovation

TECHNICAL REPORT

Computational tools for spacecraft electrostatic cleanliness and payload accomodation analysis - SPIS Science - Architectural Design Document (ADD) and Sofware Design Document (SDD)

Authors : J.-C. Mateo Velez ; P. Sarrailh ;
J. Forest (Artenum) ; B. Thiébault (Artenum) ;
B. Ruard (Artenum)

SPACE ENVIRONMENT DEPARTMENT

RT 6/17826 DESP - July 2013

BP 4025 - 2, avenue Edouard Belin
31055 Toulouse Cedex - FRANCE
Tél. : 05 62 25 25 25 - Fax : 05 62 25 25 50

SPACE ENVIRONMENT DEPARTMENT

Technical Report N° RT 6/17826 DESP

July 2013

Computational tools for spacecraft electrostatic cleanliness and payload accomodation analysis - SPIS Science - Architectural Design Document (ADD) and Sofware Design Document (SDD)

Written by :
J.-C. Mateo Velez ; P. Sarrailh ; J. Forest (Artenum) ; B. Thiébault (Artenum) ;
B. Ruard (Artenum)

Approved by :
Director
Space Environment Department
J.F. Roussel

UNCLASSIFIED
(SANS MENTION DE PROTECTION)

IDENTIFICATION CARD of ONERA REPORT N° RT 6/17826 DESP

Issued by : **SPACE ENVIRONMENT DEPARTMENT**	Contracting Agency : **ESA**	Contract Number : **4000102091/NL/AS**
	Programme card number: **201.T**	Date : **July 2013**
Title : **Computational tools for spacecraft electrostatic cleanliness and payload accomodation analysis - SPIS Science - Architectural Design Document (ADD) and Sofware Design Document (SDD)**		
Author(s) : **J.-C. Mateo Velez ; P. Sarrailh ; J. Forest ; B. Thiébault ; B. Ruard**		

SECURITY CLASSIFICATION : Civil	Timing Classification Off
Title : UNCLASSIFIED (SANS MENTION DE PROTECTION)	Title : Without object
ID Card : UNCLASSIFIED (SANS MENTION DE PROTECTION)	ID Card : Without object
Report : UNCLASSIFIED (SANS MENTION DE PROTECTION)	Report : Without object

Abstract :
This document describes the architecture of the new components developed within "Computational tools for spacecraft electrostatic cleanliness and payload accommodation analysis", here called "SPIS-SCIENCE". It is aimed at extending the capabilities of SPIS modeling framework for accurate evaluation of low-level surface electrostatic charging of science missions with low-energy plasma instruments.

As defined in the SRD [AD4], a large number of improvements will be performed in the frame of this activity, from very localized modifications/adding to the code; to very large changes of the global architecture (as e.g. instruments). These upgrades necessitate a clear definition of the interfaces between the user, the graphical user interface and the numerical kernel. This document defines the global architecture and detailed design of the new components of the SPIS-SCI software tool.

Section 2 is the global architecture design of the software.
Section 3 is the detailed design description, including the correspondence with software requirements.
Key words :

DISTRIBUTION LIST of ONERA REPORT N°RT 6/17826 DESP

Distribution of report

- **Outside ONERA :**

ESA	A. Hilgers	1 ex.

- **Inside ONERA :**

DCT/CID	Documentation	1 ex.

Distribution of identification card

- **Outside ONERA :**

- **Inside ONERA :**

Systematic distribution : DSG, DTG, DAI, DCV/ND .... 4 ex.

UNCLASSIFIED

Computational tools for spacecraft electrostatic cleanliness and payload accommodation analysis
-
SPIS-SCIENCE

Architectural Design Document (ADD)
and
Software Design Document (SDD)

June 2013

ESTEC Contract No. 4000102091/10/NL/AS

ONERA

prepared in partnership with

ARTENUM

ONERA THE FRENCH AEROSPACE LAB

ARTENUM, PARIS Science & Groupware

UNCLASSIFIED

Architectural Design Document (ADD) and Software Design Document (SDD)

June 2013

Contributors

Pierre Sarrailh
Jean-Charles Matéo-Vélez

ONERA/DESP
2 Av. Edouard Belin
31055 Toulouse cedex
France

Julien Forest
Benoit Thiébault
Benjamin Ruard

ARTENUM
24 rue Louis Blanc
75010 Paris
France

Document Status Sheet			
Document Title:			
Issue	**Date**	**Author(s) of document/change**	**Reason of change**
1.0	22/02/2013	PS, JCMV	Creation
1.1	08/03/2013	JCMV	PM4 meeting review
1.2	04/03/2013	JCMV, JF	PM4 meeting review
1.3	26/04/2013	JCMV, JF, PS	PM5 meeting review
1.4	22/05/2013	JCMV	before PM6 review
1.5	19/06/2013	JCMV	Version submitted to ESA approval
1.6	30/06/2013	JCMV	Final

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Objectives

This document describes the architecture of the new components developed within "Computational tools for spacecraft electrostatic cleanliness and payload accommodation analysis", here called "SPIS-SCIENCE". It is aimed at extending the capabilities of SPIS modeling framework for accurate evaluation of low-level surface electrostatic charging of science missions with low-energy plasma instruments.

As defined in the SRD [AD4], a large number of improvements will be performed in the frame of this activity, from very localized modifications/adding to the code; to very large changes of the global architecture (as e.g. instruments). These upgrades necessitate a clear definition of the interfaces between the user, the graphical user interface and the numerical kernel. This document defines the global architecture and detailed design of the new components of the SPIS-SCI software tool.

Section 2 is the global architecture design of the software.
Section 3 is the detailed design description, including the correspondence with software requirements.

1.2. Acronyms

ADD	Architecture Design Document
API	Application Programming Interface
BSD	Berkeley software distribution license
BT	BackTracking
CAD	Computer-aided design
CPU	Central processing unit
DF	Distribution Function
ESC	Environment and Space charge effect
ESN	Electrical Super Node
FGS	Field generated by spacecraft
FT	ForWardTracking
GEO	Geosynchronous orbit
GP	Global parameter
GPL	GNU general public license
GUI	Graphical User Interface
ICD	Interface Control Document
IME	Integrated Modelling Environment
ITO	Indium tin oxyde
IV	Current voltage sweep
LGPL	GNU Lesser General Public License
LP	Langmui Probe

MCC	Monte Carlo Collision
MEO	Middle Earth Orbit
NUM	Numerical core of SPIS
OML	Orbital Motion Limited
ONERA	Office National d'Etudes et de Recherches Aérospatiales
OSGI	Open Services Gateway initiative
PD	Particle detectors
PE	Performance and Efficiency
PIC	Particle in cell
PS	Particles from Spacecraft
RKCK	Runge-Kutta Cash-Karpe
SEEE	Secondary electron emission by electron impact
SP	Software processing
SPS	Spherical Plasma Sensor
S/C	Space craft
SPIS	Spacecraft Plasma Interaction Software
SPIS-CORE	Current SPIS development branch available on spis.org website
SPIS-GEO	Simplified MEO/GEO tools for spacecraft charging, ESA Co4000101174
SPIS-SCI	SPIS-SCIENCE: Computational tools for spacecraft electrostatic cleanliness and payload accomodation analysis
SRD	Software Requirements Document
STEP	Standard for the Exchange of Product model data (ISO 10303)
STG	Semi-Transparent Grid
SVN	SubVersion control
TP	Test Particle
UI	User Interface of SPIS
UML	Unified Modeling Language
UR	User Requirement
PS	Plasma Sensor
SRD	Software Requirements Document
SWA	Solar Wind Analyser
VPD	Virtual Particle Detector
VPS	(Virtual) plasma sensor
VSM	Virtual surface mesh
wrt	with respect to

1.3. Applicable Documents

[ITT] Invitation to tender – AO/1-6368/10/NL/AF - Computational tools for spacecraft electrostatic cleanliness and payload accommodation analysis

[PROP] Proposal for a Computational tools for spacecraft electrostatic cleanliness and payload accommodation analysis, in response to AO/1-6368/10/NL/AF, ONERA, ARTENUM, IRF, IRAP.

[AD1] This issue, User Requirements Document, version 2.1, ESA-SPIS-SCI-URD-2011-03-01-2.1.

[AD2] SPIS-GEO User Requirements Document version 1.3, ESA-SPISGEO-D1-URD-2010-09-002, ESA contract 4000101174.

[AD3] SPIS-GEO Software Requirements Documents version 1.1, ESA-SPISGEO-D3-SRD-2010-11-001, ESA contract 4000101174.

[AD4] This issue, Software Requirements Documents, version 2.1.

[AD5] This issue, Architecture Design Document, version 1.2, June 2011.

1.4. Reference Documents

[SPINE WS] 17th SPINE meeting (documents, presentations) :

http://dev.spis.org/projects/spine/home/meeting/mxvii

[RD1] ESA PSS-05-0 Issue 2 (February 1991) ESA Software Engineering Standards.

[RD2] jHepWork's Web page, http://jwork.org/jhepwork/index.php

[RD3] VisAD'Web page, http://www.ssec.wisc.edu/~billh/visad.html

[RD4] Xstream's Web page, http://xstream.codehaus.org/

[RD5] JFreeChart's Web page, http://www.jfree.org/jfreechart/

[RD6] Spacecraft plasma interaction analysis and simulation toolkit, Final Report, ESTEC Co 16806/02/NL/JA

[RD7] Time Dependent simulator of charge and discharge on spacecraft.

2. ARCHITECTURE DESIGN

This section describes the global architecture of the SPIS software at the end of this activity. In 2.1, we present the initial model and architecture. Paragraph 2.2 is the overview of the main new functionalities developed. In 2.3, the architecture of new or modified modules. Their interfaces with external tools are described in 2.4.

2.1. Background

The main purpose of SPIS 4.3 and previous versions was to model interactions between a spacecraft and its surrounding environment in term of surface charging. As a large set of situations can possibly be met in flight, the structure of the code was developed in a highly modular and flexible fashion [FR-SPIS12-10-2007]. This approach facilitated the communication and usage of the code. That choice conducted to the choice of an open source and object oriented language (JAVA/jython).

SPIS can be considered as a modular simulation structure for which modules can be added. The code can be split in two parts, called SPIS-Num for the numerical core and SPIS-UI for the user interface. The main physical processes modeled in the SPIS version used at the beginning of this activity are summarized in next figures. On the solvers side (SPIS-Num) we dispose at the start of this activity of a modular flexible structure [RD6-RD7]. All types of objects under use follow a generic model (typically an abstract class), which allows defining new versions that can be integrated without further work. This can be done for particle populations (including solvers), particle sources, field solvers, environment, etc. Java introspection capabilities even allow integrating a new version of some types without changing the existing code, only by switching to the new version by typing the new version name in the GUI (like a plug-in). The implemented list of solvers, which can easily be extended, is the following.

Concerning matter, the main models are a Particle-In-Cell (PIC) solver for a Monte Carlo solving of Vlasov equation, and a Boltzmann distribution to describe the thermal equilibrium distribution of electrons. Particle source distributions include a whole library. Gas phase collisions can be modelled through a MCC method, with only charge exchange reactions implemented as of today. Available surface interaction models cover most interactions relevant to space environment: secondary electron emissions (under electron or proton), photo emission, radiation-induced-conductivity, erosion. A complex 3D multizone model was defined to handle simultaneously in a single simulation a dense quasi neutral region and a space charge region, typically influenced by an unscreened positive potential nearby.

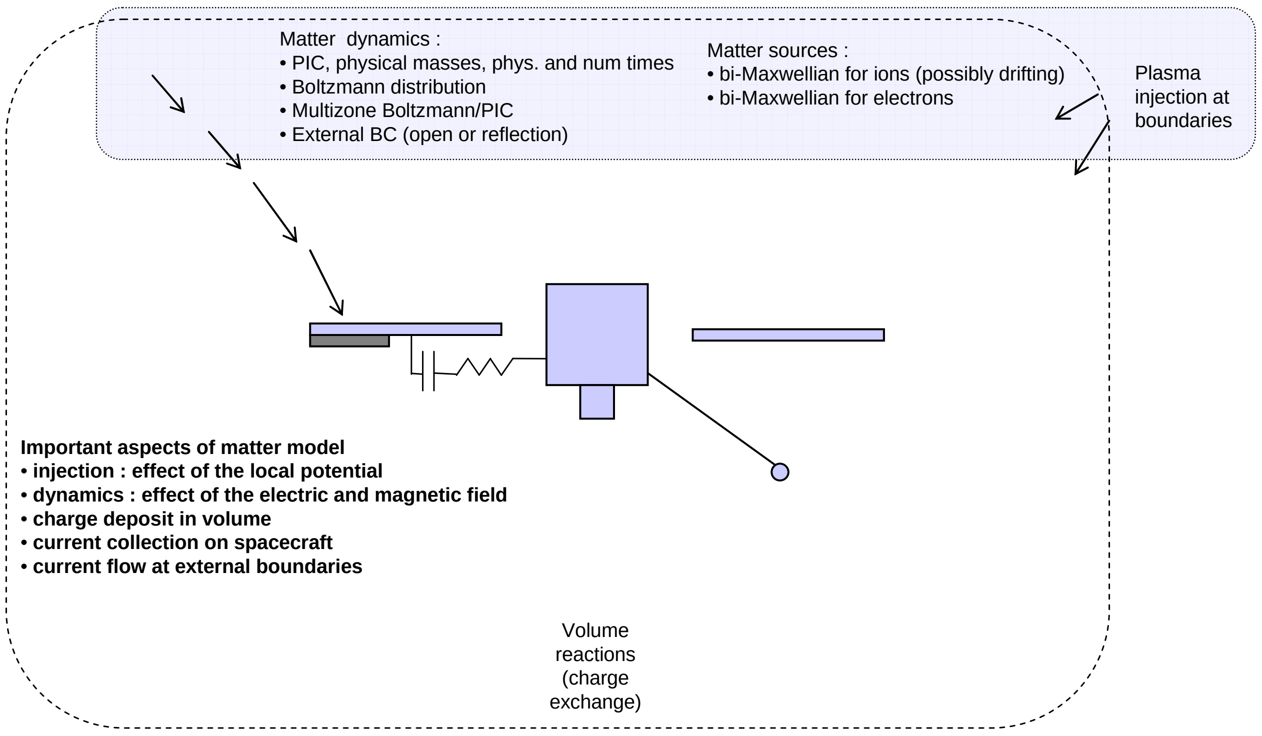

Matter dynamics :
• PIC, physical masses, phys. and num times
• Boltzmann distribution
• Multizone Boltzmann/PIC
• External BC (open or reflection)
Matter sources :
• bi-Maxwellian for ions (possibly drifting)
• bi-Maxwellian for electrons
Plasma injection at boundaries
Important aspects of matter model
• injection : effect of the local potential
• dynamics : effect of the electric and magnetic field
• charge deposit in volume
• current collection on spacecraft
• current flow at external boundaries
Volume reactions (charge exchange)

Figure 1 - Matter model in SPIS 4.3 version

Concerning fields, the Poisson equation finite element solver follows a pre-conditioned conjugate gradient method. The boundary conditions can either be Dirichlet, Neumann or a mix of them (known as Robin or Fourier), which allows a better modelling of pre-sheath conditions ($1/r^n$ behaviour). Non linear Poisson equation (i.e. Poisson including Boltzmann distributions for the electrons) can also be solved with an implicit method, offering stability even for a Debye length smaller than cells. Specific features were added to handle singular geometries, either very thin wires (of which radius cannot be meshed without generating degenerated elements) or very thin plates (of which edges cannot be meshed). The method consists in analytically subtracting the singular part of the potential field resulting from the geometrical singularity, and solving the regular part (this splitting being of course transparent for the user).

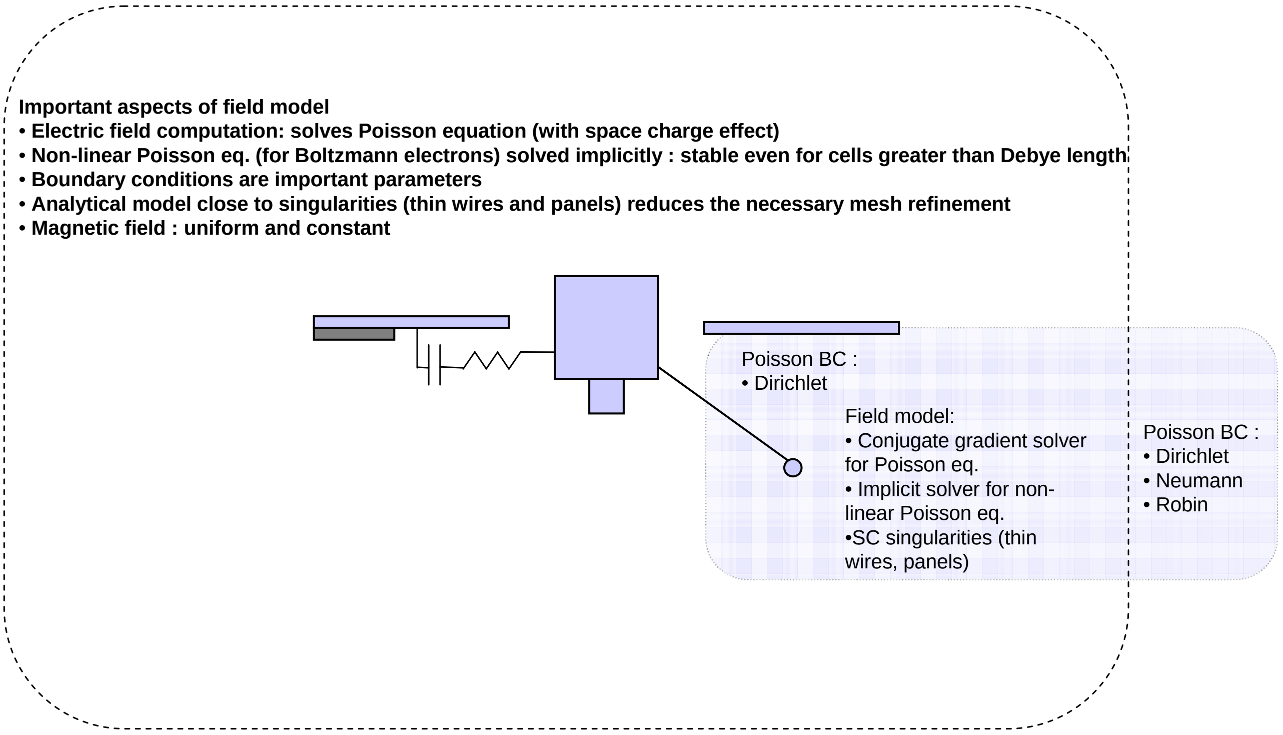

Important aspects of field model
• Electric field computation: solves Poisson equation (with space charge effect)
• Non-linear Poisson eq. (for Boltzmann electrons) solved implicitly : stable even for cells greater than Debye length
• Boundary conditions are important parameters
• Analytical model close to singularities (thin wires and panels) reduces the necessary mesh refinement
• Magnetic field : uniform and constant
Poisson BC : • Dirichlet
Field model: • Conjugate gradient solver for Poisson eq. • Implicit solver for non-linear Poisson eq. •SC singularities (thin wires, panels)
Poisson BC : • Dirichlet • Neumann • Robin

Figure 2 - Field model in SPIS 4.3 version

Surface interactions with plasma considered in SPIS are the secondary electron emission under electron and photon impact. Yields are calculated automatically within SPIS pending on macroscopic material properties defined at user level. For instance the yield of true electron emission under electron impact is computed using the maximal yield and the energy of the maximum. Distribution functions are Maxwellian in volume (or Lambertian in surface). Secondaries from secondaries is activated for electrons from environment and artificial sources on spacecraft. Artificial sources models are maxwellian (possibly with a drift), axisymmetric and two axis symmetry (plus sources like field effect emission). Erosion model is applicable to Xenon ions.

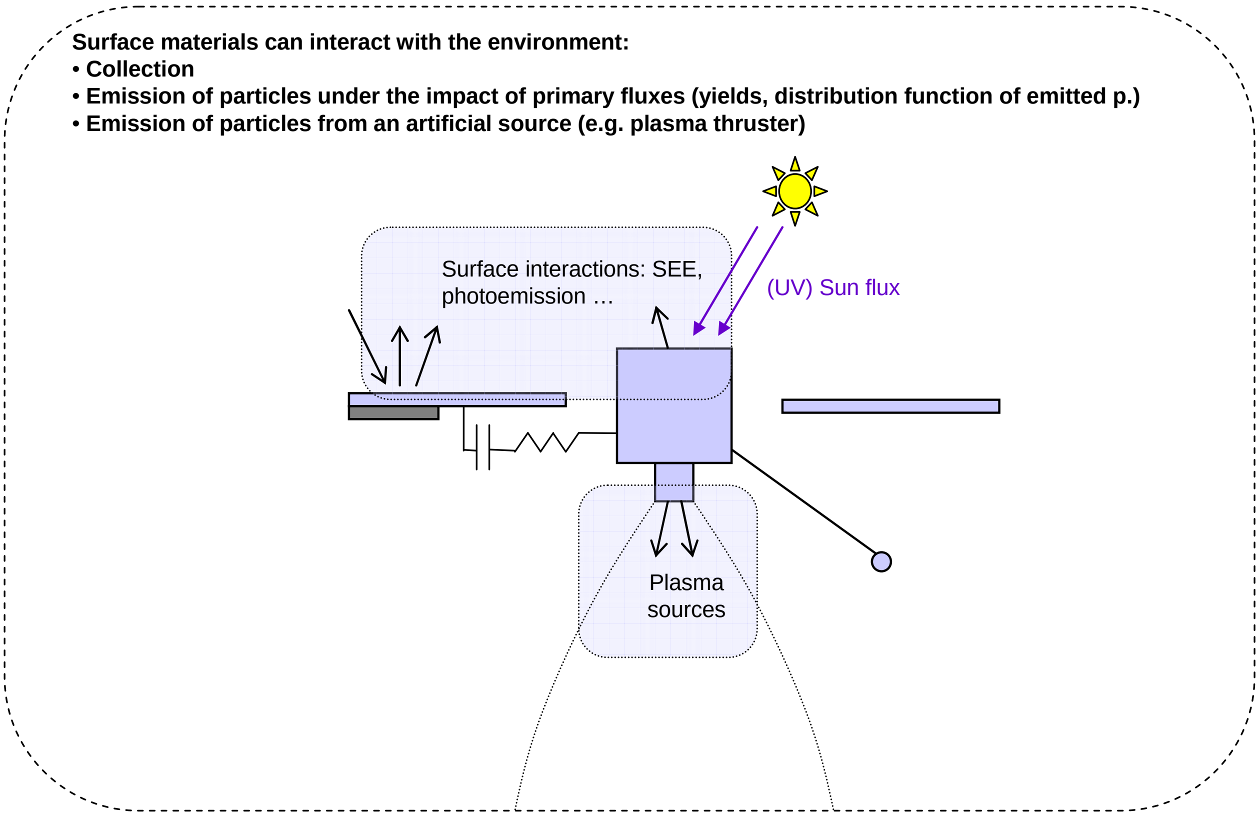

Surface materials can interact with the environment:
• Collection
• Emission of particles under the impact of primary fluxes (yields, distribution function of emitted p.)
• Emission of particles from an artificial source (e.g. plasma thruster)
Surface interactions: SEE, photoemission …
(UV) Sun flux
Plasma sources

Figure 3 - Spacecraft surface interaction with environment in SPIS 4.3 version

The evolution of potentials on the spacecraft involves an equivalent circuit taking into account the coating capacitances and conductances (surface and volume conductivities), and also user-defined "discrete" components (extra resistors, biases or capacitors added between subsystems). A spacecraft circuit implicit solver was designed to handle problems with very different time scales thanks to physics predictors of the plasma current variations in reaction to potential changes.

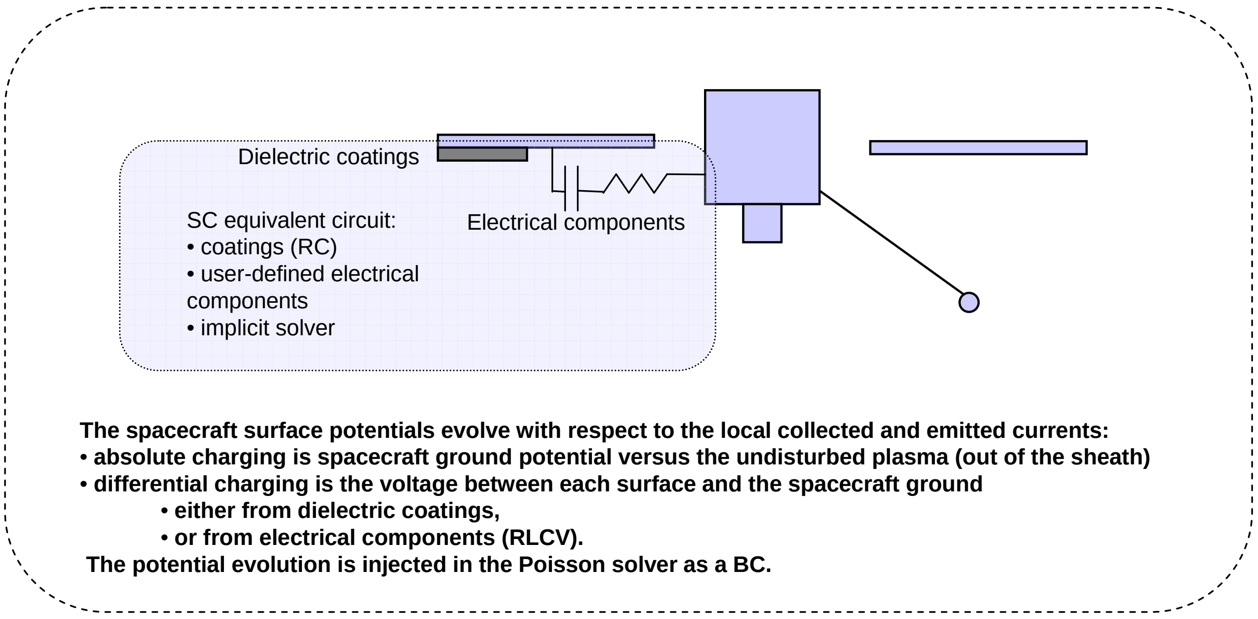

Dielectric coatings
SC equivalent circuit:
• coatings (RC)
• user-defined electrical components
• implicit solver
Electrical components
The spacecraft surface potentials evolve with respect to the local collected and emitted currents:
• absolute charging is spacecraft ground potential versus the undisturbed plasma (out of the sheath)
• differential charging is the voltage between each surface and the spacecraft ground
• either from dielectric coatings,
• or from electrical components (RLCV).
The potential evolution is injected in the Poisson solver as a BC.

Figure 4 - Surface potential model in SPIS 4.3 version

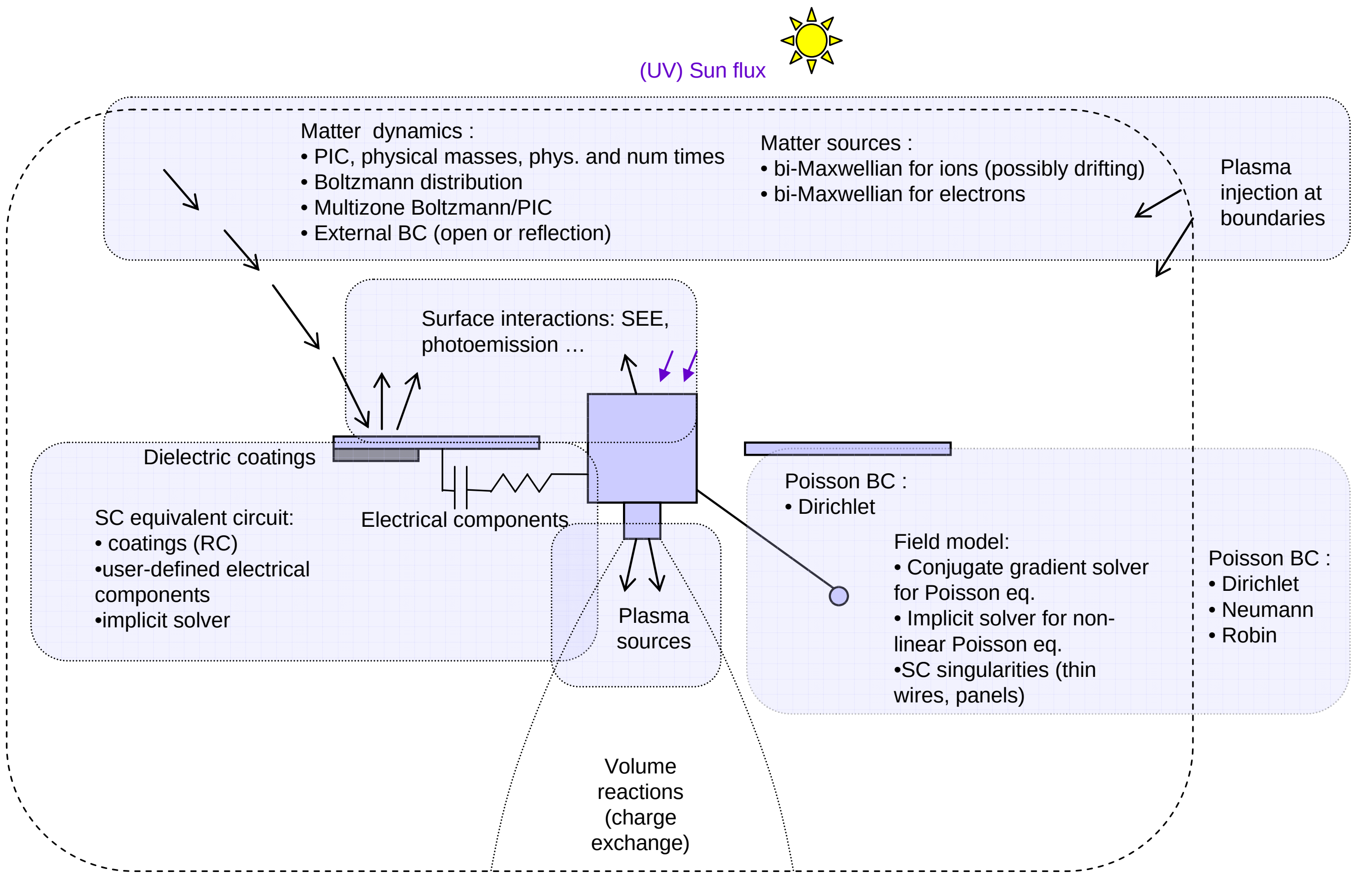

(UV) Sun flux
Matter dynamics :
• PIC, physical masses, phys. and num times
• Boltzmann distribution
• Multizone Boltzmann/PIC
• External BC (open or reflection)
Matter sources :
• bi-Maxwellian for ions (possibly drifting)
• bi-Maxwellian for electrons
Plasma injection at boundaries
Surface interactions: SEE, photoemission …
Dielectric coatings
SC equivalent circuit:
• coatings (RC)
•user-defined electrical components
•implicit solver
Electrical components
Plasma sources
Poisson BC :
• Dirichlet
Field model:
• Conjugate gradient solver for Poisson eq.
• Implicit solver for non-linear Poisson eq.
•SC singularities (thin wires, panels)
Poisson BC :
• Dirichlet
• Neumann
• Robin
Volume reactions (charge exchange)

Figure 5 - Global spacecraft plasma interaction model in SPIS 4.3

The numerical algorithm consists in imbricate loops of calculations:

- Matter dynamics
  - Each population k is injected, moved, collected on a given duration popDuration(k), using a time step popDt(k)
- Field/Matter coupling
  - Poisson equation is solved with the particles space charge
  - This loop is repeated until plasmaDuration time is reached (with plasmaDt time steps)
- Spacecraft level
  - Computation of the collected current during the time step simulationDt
  - Computation of the emitted current (sources, interactions) during the time step simulationDt
  - Surface potential evolution during simulationDt are computed with the circuit solver (conductance, capacitance and inductance model)
  - The simulation stops when the time reaches duration

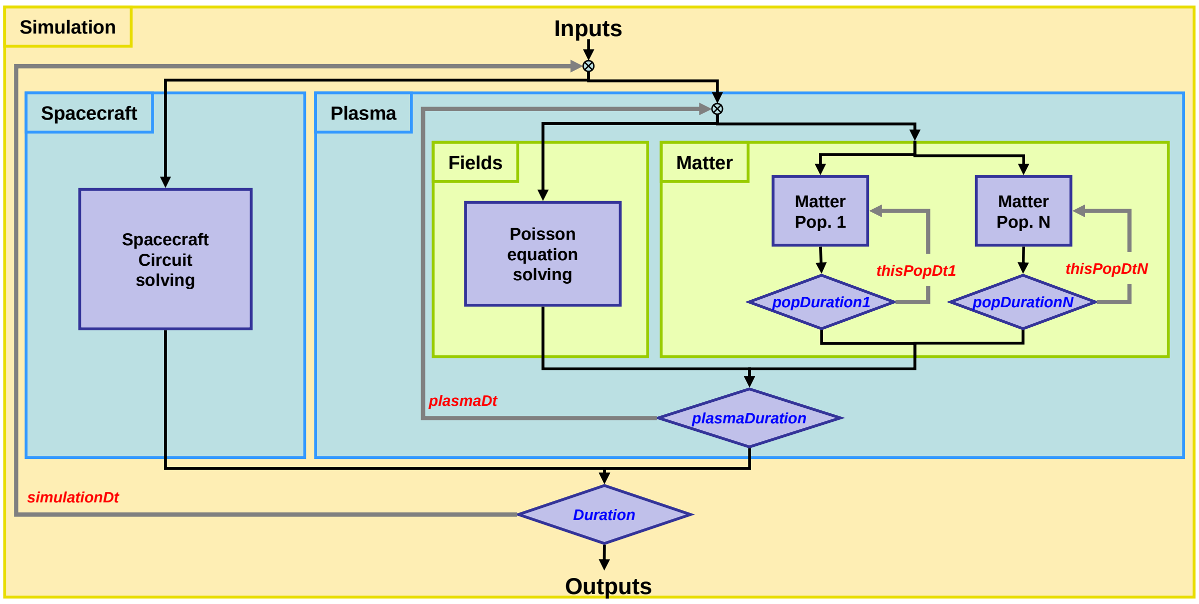


Figure 6 - Numerical loops in SPIS

The user (and software) interface is a real simulation framework, with the capability to handle simulation data (CAD objects, mesh, groups etc. and data fields living on them, for pre processing or post processing) and handle tasks chaining thanks to a task manager. Following the open source logics, as many external elements as possible were integrated or interfaced with SPIS. The modelling and meshing are performed by the external tool GMSH. Three-dimensional visualisations make use of VTK scientific visualisation toolkit (and Cassandra viewer internally, or Paraview externally). Many other open source tools are also used (JFreeChart for 2D plotting, Jython, a Java-based Python interpreter, for the script console, etc.).

The interfacing between the SPIS-UI framework and the numerical kernel SPIS-NUM, is presented in details in [RD2]. SPIS-UI is a modular framework, where each functional module is embedded under the form of standardized containers called Tasks. This includes SPIS-NUM which is called through the JyTop and SpisNumCaller classes, themselves called through different Tasks. Scheme 1, here-after shows the control process between the calling tasks, on the left side, and the low level components like SPIS-NUM, on the right side, through the different embedding classes. The elements written in blue are methods or objects of SPIS-NUM. The list given in ANNEXE-1 completes this scheme and gives the correspondence between most of the tasks and the low level components and business codes.

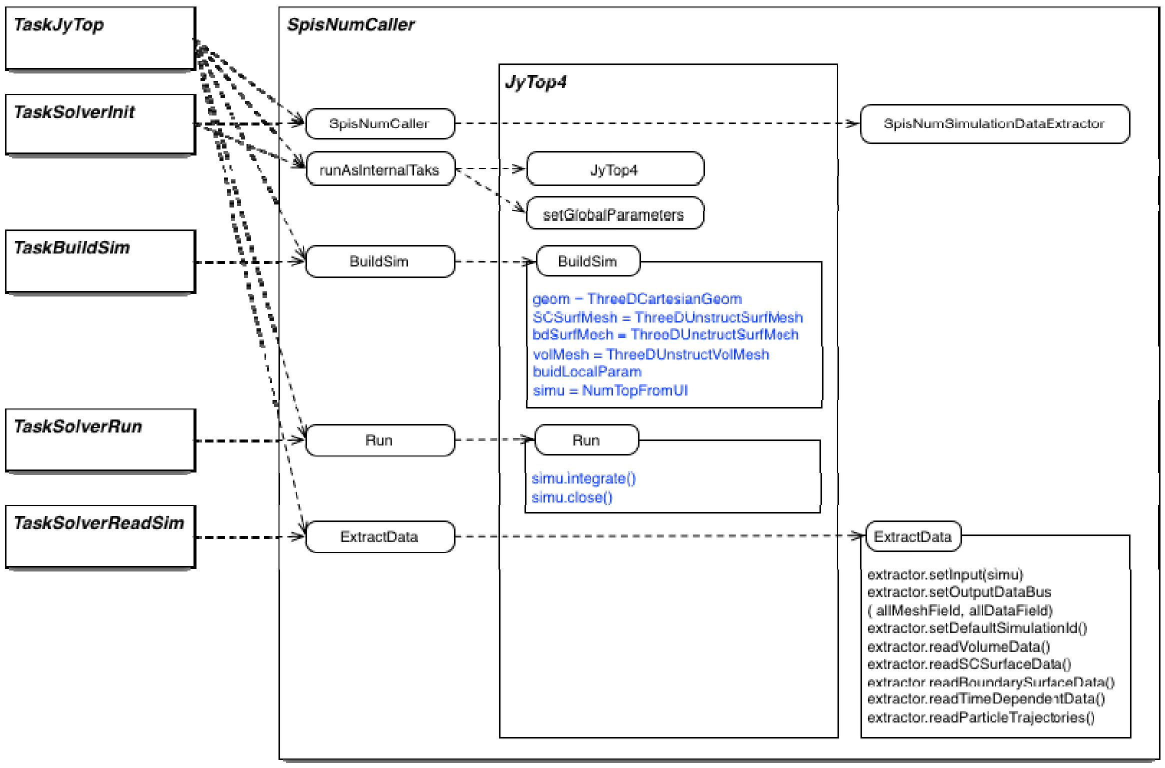

TaskJyTop
TaskSolverInit
TaskBuildSim
TaskSolverRun
TaskSolverReadSim
SpisNumCaller
JyTop4
SpisNumCaller
runAsInternalTaks
JyTop4
setGlobalParameters
SpisNumSimulationDataExtractor
BuildSim
BuildSim
geom − ThreeDCartesianGeom
SCSurfMesh = ThreeDUnstructSurfMesh
bdSurfMech = ThreeDUnctructSurfMech
volMesh = ThreeDUnstructVolMesh
buidLocalParam
simu = NumTopFromUI
Run
Run
simu.integrate()
simu.close()
ExtractData
ExtractData
extractor.setInput(simu)
extractor.setOutputDataBus
(allMeshField, allDataField)
extractor.setDefaultSimulationId()
extractor.readVolumeData()
extractor.readSCSurfaceData()
extractor.readBoundarySurfaceData()
extractor.readTimeDependentData()
extractor.readParticleTrajectories()

Figure 7 - Current control of SPIS-NUM from SPIS-UI

The TaskJyTop task performs in one-click the equivalent of the TaskSolverInit, TaskbuildSim, TaskSolverRun and TaskSolverRead, which corresponds to the whole simulation process.
Inputs data needed by SPIS-NUM are provided under the form of:

- A set of three meshes (computational volume, S/C surface mesh, external boundary mesh);
- A set of fields deployed on the meshes, i.e DataFields converted as arrays;
- A set of global parameters;
- NASCAP based material descriptions.

The DataField and mesh conversion, including the elements renumbering, are performed in the TaskSpisNumInterface task, that itself call the SpisNumInterface class. Most of the mesh structure conversion and extraction (e.g extraction of the surface mesh corresponding to the S/C surface) are performed by the Penelope library (previously called JFreeMesh). All converted data (meshes) are stored in the sharedNum['SNMesh'] shared dictionary, itself used by JyTop4. Local parameters and material properties are converted directly into the JyTop4 class. All parameters are passed under the form of “simple” structures, i.e. arrays of local parameters or key-value set for global parameter.

The results are extracted from SPIS-NUM through the SpisNumSimulationDataExtractor. Fields deployed on the grid are converter into DataFields.

This scheme shows the limitations of the present implementation of the SPIS-UI / SPIS-NUM interface. Most of the settings are passed to SPIS-NUM through its constructor. In this context, it is quite difficult to adjust or restart the simulation and/or finely control it from the SPIS-UI. There is no possibility of command or triggering during the simulation running (i.e. integrate() function of SPIS-NUM) from SPIS-UI. Reciprocally, there is no notification mechanism for SPIS-NUM to SPIS-UI to inform this last one of the progress status of the simulation and/or a key event like the validation of produced data. Last, in the current implementation, if the data extraction can be done on-demand from SPIS-UI, only data already previously generated by SPIS-NUM and will not force this one to update these results. For instance, if the period to generation of output (e.g. plasmaPotMapMonitorStep) is higher than the current simulation time no output will be generated and could be recovered by the data extraction module. On the other hand, all generated data are extracted at the end of the simulation in the same time and loaded in memory, leading sometimes to a memory overflow at the end of the simulation. These elements present a severe limitation for scientific application, where simulations can be very large and long and/or where numerous data should be extracted at a given frequency ($T_{observation}$).

2.2. Main new components of SPIS-SCIENCE

According the previous analysis and [SRD], the main components that have been modified or added to the structure are:

- **Instruments** modules aiming at measuring particle spectrograms and plasma characteristics within the course of the simulations
- **Semi-Transparent Grids** to mimic scientific instrumentation.
- **Non-maxwellian distribution functions** for ambient an secondary populations
- **Thin elements** such as wires and solar panels
- **Performance and accuracy** increase
- Extension of **material properties** format (tables)
- Extended **control of the Num integration by UI** in order to interact more easily with the user
- Development of **Transitions** aiming at simulating transient phase in some typical situations.

The main new or modified components of SPIS-SCIENCE are schematically represented in Figure 8 and Figure 9.

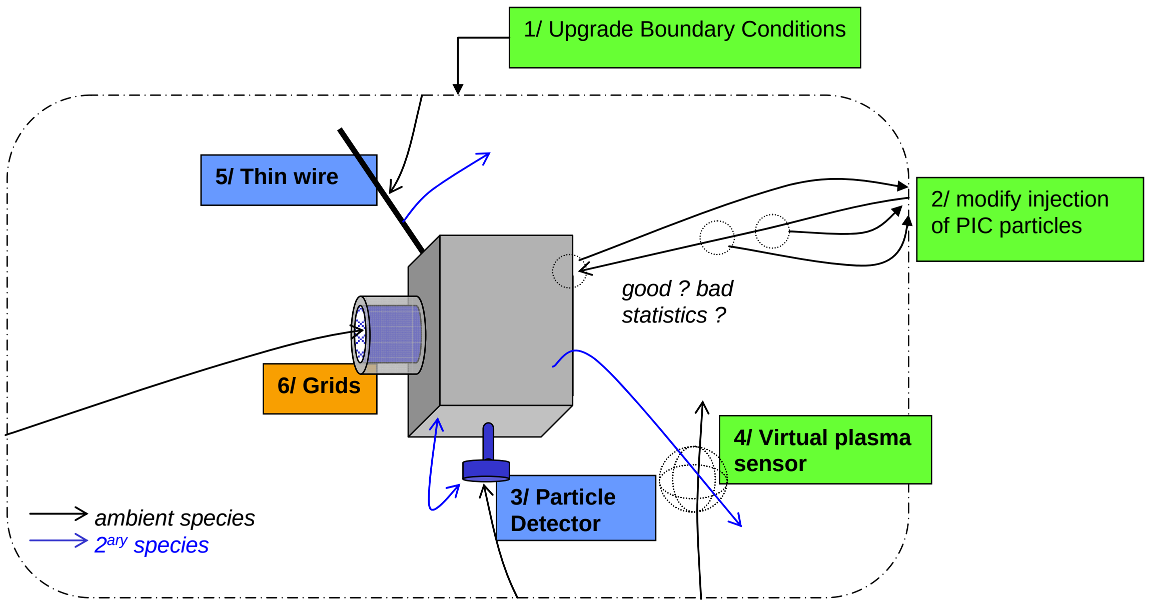


Figure 8 - Main new or modified components of SPIS-SCIENCE (1/2)

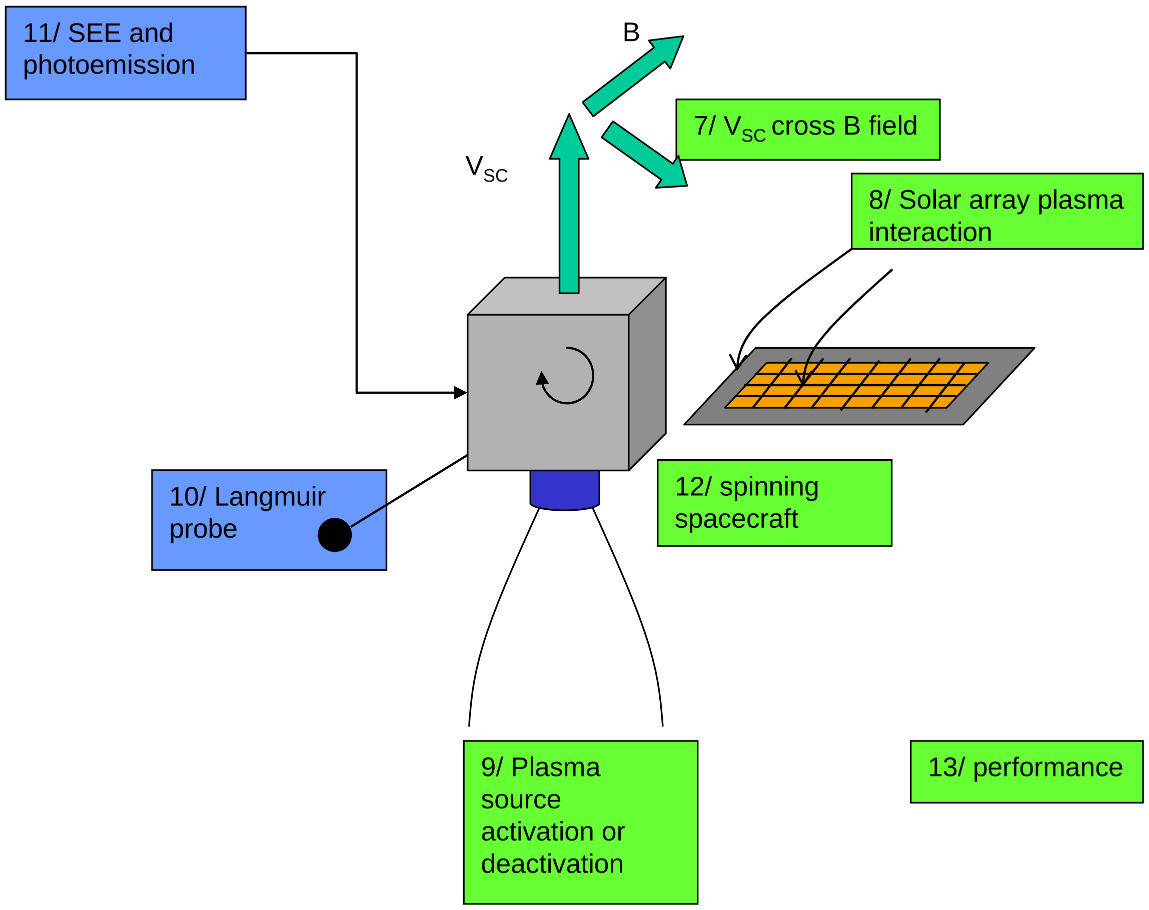

11/ SEE and photoemission
B
7/ V_{SC} cross B field
V_{SC}
8/ Solar array plasma interaction
10/ Langmuir probe
12/ spinning spacecraft
9/ Plasma source activation or deactivation
13/ performance

Figure 9 - Main new or modified components of SPIS-SCIENCE (2/2)

2.3. Architecture of SPIS components

A large number of packages and classes has been modified or added to the previous SPIS versions. In conjunction with SPIS-GEO activity, the framework has been deeply refactored and the UI2NUM phase has been significantly improved. The numerical kernel was also largely improved and increased to answer the user and software requirements.

In this section, we present the full list of packages of the new SPIS version.

2.3.1. Overview

SPIS-SCI is fully based on the current stable version of the main branch of SPIS in its version 5.0, as developed in the frame of SPIS-GEO/MEO project. As for the previous versions (4.3 and earlier, also called Legacy), SPIS is divided into main layers:

- o SPIS-NUM, which corresponds to the simulation kernel;
- o SPIS-UI, which corresponds to the Integrated Modelling Environment (IME) to drive the whole modelling process and support the dedicated GUI.

In the frame of the SPIS-SCI project, both layers have been updated and extended to integrate the additional functions and capabilities requested by the objectives of the project. For both layers, the developments done here have been reversed into the main development branch of SPIS.

2.3.2. SPIS-NUM

The numerical kernel block diagram of the next figure presents the SPIS-NUM packages and their main interactions. The top level class called by UI is NumTopFromUI and is constituted by a simulation, itself gathering a spacecraft object, a plasma and possibly extra objects such as grids and transitions. The volume (VOL) package is composed of classes relative to volumic meshes, data fields living on this mesh, population distributions and interactions. The plasma object is composed of volume distributions associated to a Poisson solver, located in the Solver package. This package also provides Matter classes to move particles of the Volume package. The spacecraft is composed of a circuit solver located in the Circ package. The circuit solver also takes account of semi-transparent grid objects. Both spacecraft and grids are associated to surface interactions of the SURF package (collection, emission of particles). The particle pusher of the Matter solver moves particles on the volume mesh and mark particles arriving on spacecraft and grid surface meshes. Then, the surface interactions generate secondary surface distributions used as injection condition of volume distributions. The Transition classes modify the simulation object and its components during the time integration. Finally, the UTIL package collects a large number of methods (sampling, mathematics, etc…) used in all packages.

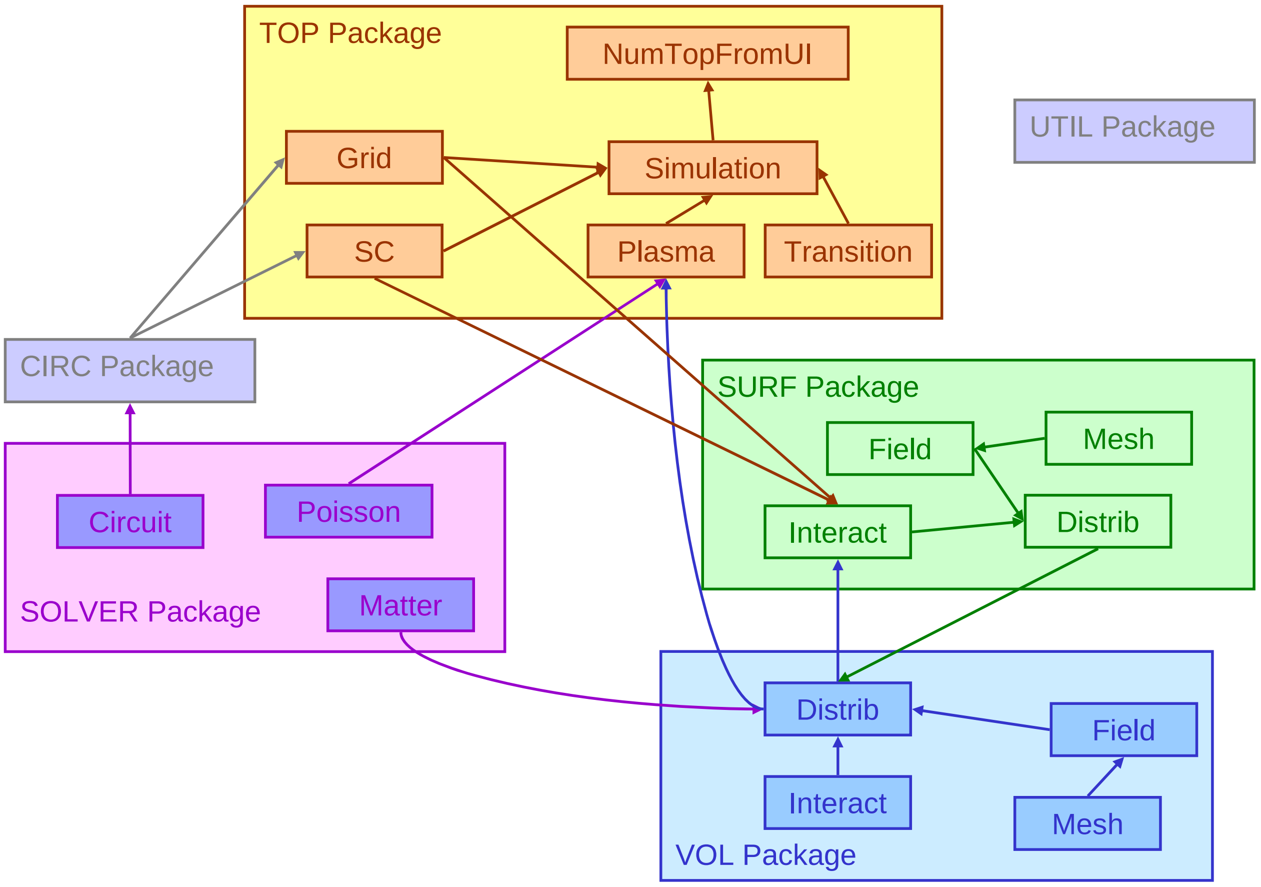


Figure 10 - Block diagram of SPIS-NUM packages, with main interactions between them

For a complete understanding of SPIS-NUM classes hierarchy and interactions, a proper documentation can be generated using the Javadoc command of java. In the hierarchy tree below, we have highlighted of green and blue the components developed or modified in the frame of the SPIS-SCIENCE and SPIS-GEO activities respectively.

```
java.lang.Object
   spis.Util.Func.Abs (implements spis.Util.Func.ScalFunctionOfScal)
   spis.Surf.SurfField.AbstractCurrentScaler (implements spis.Surf.SurfField.CurrentScaler)
      spis.Surf.SurfField.BarrierCurrentScaler
         spis.Surf.SurfField.VariableBarrierCurrentScaler
         spis.Surf.SurfField.VariableBarrierCurrentScaler2
            spis.Surf.SurfField.AutomaticBarrierCurrentScaler
      spis.Surf.SurfField.GlobalTempCurrentScaler
         spis.Surf.SurfField.OMLCurrentScaler
         spis.Surf.SurfField.OMLCurrentScalerWithNeutrality
         spis.Surf.SurfField.OMLCurrentScalerWithNeutralityInf
         spis.Surf.SurfField.SecondaryRecollectionCurrentScaler
         spis.Surf.SurfField.SmoothedGlobalTempCurrentScaler
      spis.Surf.SurfField.LocalTempCurrentScaler
         spis.Surf.SurfField.LocalOMLCurrentScalerWithNeutrality
      spis.Surf.SurfField.NoVariationCurrentScaler
      spis.Surf.SurfField.OldBarrierCurrentScaler
```

```
    org.spis.instruments.AbstractInstrument<MeasureType,ParameterType> (implements org.spis.instruments.Instrument<MeasureType,ParameterType>)
      spis.Util.Instrument.TopInstrument<T>
        spis.Util.Instrument.ParticleDetector
          spis.Util.Instrument.LangmuirProbe
        spis.Util.Instrument.PlasmaSensor<T>
          spis.Util.Instrument.AvSCPotentialMonitor
          spis.Util.Instrument.ESNCurrentMonitor
          spis.Util.Instrument.ESNIndividualCurrentMonitor
          spis.Util.Instrument.ESNPotentialDifferenceMonitor
          spis.Util.Instrument.ESNPotentialMonitor
          spis.Util.Instrument.ESNPotentialVariationMonitor
          spis.Util.Instrument.LinePS<T>
            spis.Util.Instrument.PotentialLPS
          spis.Util.Instrument.PointPS<T>
            spis.Util.Instrument.DensityPS
            spis.Util.Instrument.PotentialPS
            spis.Util.Instrument.SphericalPS<T>
              spis.Util.Instrument.EnergyDistFuncPS
                spis.Util.Instrument.TotalEnergyDistFuncPS
              spis.Util.Instrument.KineticEnergyPS
              spis.Util.Instrument.VelocityDistFunctionPS
                spis.Util.Instrument.SurfaceFluxDistFunctionPS
            spis.Util.Instrument.VolDistribMomentPS
          spis.Util.Instrument.SimulationTimeStepMonitor
          spis.Util.Instrument.TotalCurrentOnSC
          spis.Util.Instrument.TotalEnergyPS
          spis.Util.Instrument.TotalSuperParticlePS
        spis.Util.Instrument.VirtualInstrument<T>
          spis.Util.Instrument.VirtualParticleDetector
    spis.Util.Func.And (implements spis.Util.Func.TestOfInt)
    spis.Util.Vect.Array
    spis.Solver.Matter.BasicPusher (implements spis.Solver.Matter.ParticlePusher)
    spis.Vol.BC.BC
      spis.Vol.BC.PoissonBC
        spis.Vol.BC.DirichletPoissonBC
        spis.Vol.BC.FourierPoissonBC
        spis.Vol.BC.MixedDirichletFourierPoissonBC
    spis.Vol.VolMesh.Boundary
    spis.Solver.Poisson.BoundaryElecFieldModel (implements spis.Solver.Poisson.ElecFieldModel)
    spis.Vol.VolField.BoundaryField
    spis.Util.Func.BoundedFunctionOf3Scal (implements spis.Util.Func.ScalFunctionOf3Scal)
    spis.Vol.VolMesh.Centring (implements java.io.Serializable)
    spis.Circ.Circ.Circ (implements spis.Circ.Circ.CircIf)
    spis.Circ.CircField.CircField
      spis.Circ.CircField.DirCircField
    spis.Solver.Circuit.CircSolve
      spis.Solver.Circuit.CircSolveTestImplicit
    spis.Solver.Circuit.CircSolveDouble
    spis.Solver.Circuit.CircSolveTest
    spis.Util.io.CircuitReader
    spis.Util.Func.CombinationOfTwoScalFunctionOfScal (implements spis.Util.Func.ScalFunctionOfScal)
    spis.Top.Default.Common
    spis.Solver.Matter.ComplexPusher (implements spis.Solver.Matter.ParticlePusher)
    spis.Surf.SurfField.ConstantCurrent (implements spis.Surf.SurfField.CurrentScaler)
```

- spis.Util.Func.ConstantFunction (implements spis.Util.Func.ScalFunctionOfNothing, spis.Util.Func.ScalFunctionOfScal)
- spis.Surf.SurfInteract.ConstantFunctionOfDistrib (implements spis.Surf.SurfInteract.FuncOfDistrib)
- spis.Util.List.ConstantSampler (implements spis.Util.List.OneDSamplable)
- spis.Util.Func.ConvertibleUnitScalFunctionOfScal (implements spis.Util.Func.ScalFunctionOfScal)
  - spis.Util.Func.Cos
  - spis.Util.Func.Exp
  - spis.Util.Func.ExpPlusConst
  - spis.Util.Func.Heaviside
  - spis.Util.Func.IsEqual (implements spis.Util.Func.TestOfInt)
  - spis.Util.Func.LimitedExp
  - spis.Util.Func.Log
  - spis.Util.Func.Max
  - spis.Util.Func.Min
  - spis.Util.Func.OMLFactor
  - spis.Util.Func.Sin
  - spis.Util.Func.VariableTeLaw
- spis.Vol.Geom.CoordinateSystemTools
  - spis.Vol.Geom.CartesianSystem
  - spis.Vol.Geom.CylindricalSystem
  - spis.Vol.Geom.SphericalSystem
- spis.Util.Func.CosMultiD (implements spis.Util.Func.ScalFunctionOfVect)
- spis.Util.Instrument.CreateDefaultMonitoringInstruments
- spis.Util.Monitor.DensityMonitor (implements spis.Util.Monitor.Monitorable)
- spis.Surf.SurfInteract.DeprecatedModelParamSet
- spis.Circ.Circ.DIDV
  - spis.Circ.Circ.DIDVfromSurfDIDV
  - spis.Circ.Circ.MultipleDIDV
  - spis.Circ.Circ.reducableDIDV
    - spis.Circ.Circ.RedDIDVfromRegDIDV
- spis.Util.Phys.DimScal (implements java.io.Serializable)
  - spis.Util.Phys.ScalableScalar
- spis.Surf.SurfInteract.DiscretisedFunctionOfParticleEDeprecated
- spis.Util.Instrument.DistFuncOfOneVariable (implements java.io.Serializable)
- spis.Util.DistribFunc.DistributionFunction
  - spis.Util.DistribFunc.TabulatedDistributionFunction
    - spis.Util.DistribFunc.IsotropicTabulatedDistributionFunction (implements spis.Util.DistribFunc.IsotropicInterface)
      - spis.Util.DistribFunc.IsotropicBiMaxwellianDF
      - spis.Util.DistribFunc.IsotropicKappaDF1
      - spis.Util.DistribFunc.IsotropicMaxwellianDF
    - spis.Util.DistribFunc.MaxwellianDF
- spis.Util.Func.DistToPoint (implements spis.Util.Func.ScalFunctionOfVect)
- spis.Util.Func.DriftingMaxwellian3VFunction (implements spis.Util.Func.ScalFunctionOf3Scal)
- spis.Vol.VolField.EField
  - spis.Vol.VolField.PotEField
- spis.Util.io.ElecCircuitReader
- spis.Circ.Circ.ElecComponent
- spis.Util.Monitor.ElecSuperNodeMonitor
- spis.Top.Plasma.Environment
  - spis.Top.Plasma.BiMaxwellianEnvironment
  - spis.Top.Plasma.ExtendedEnvironment
- spis.Util.Monitor.FieldMonitor
- spis.Surf.SurfInteract.FunctionOfDistrib (implements spis.Surf.SurfInteract.FuncOfDistrib)
- spis.Surf.SurfInteract.FunctionOfDistribFromFuncOfPart (implements spis.Surf.SurfInteract.FuncOfDistrib)
- spis.Surf.SurfInteract.FunctionOfParticle

```
      spis.Surf.SurfInteract.FunctionOfParticleE (implements spis.Util.Func.ScalFunctionOf4Scal, spis.Util.Func.ScalFunctionOfScal)
         spis.Surf.SurfInteract.EnergyFunction
         spis.Surf.SurfInteract.InvertableFunctionOfParticleE (implements spis.Util.Func.ReverseUnitCheckable)
            spis.Surf.SurfInteract.RangeFunction
               spis.Surf.SurfInteract.DoublePowerLawRangeFunction
               spis.Surf.SurfInteract.ProtonRangeFunction1
         spis.Surf.SurfInteract.ReciprocalOfFunctionOfParticleE
         spis.Surf.SurfInteract.TransmissionFunction
      spis.Surf.SurfInteract.FunctionOfParticleETheta (implements spis.Util.Func.ScalFunctionOf2Scal, spis.Util.Func.ScalFunctionOf4Scal, spis.Util.Func.ScalFunctionOf5Scal, spis.Util.Func.ScalFunctionOfScal)
         spis.Surf.SurfInteract.CompositeFunctionOfParticleETheta
         spis.Surf.SurfInteract.DampedEnergyFunction
         spis.Surf.SurfInteract.ElecBackscatterFunction
            spis.Surf.SurfInteract.ExtendedElecBackscatterFunction
         spis.Surf.SurfInteract.GRBOErosionYield
         spis.Surf.SurfInteract.SEEEYieldFunction1
            spis.Surf.SurfInteract.ExtendedSEEEYieldFunction1
         spis.Surf.SurfInteract.SEEPYieldFunction1
      spis.Surf.SurfInteract.FunctionOfParticleNothing (implements spis.Util.Func.ScalFunctionOf3Scal, spis.Util.Func.ScalFunctionOfNothing)
         spis.Surf.SurfInteract.FunctionOfMaterial
   spis.Util.Func.GFunction (implements spis.Util.Func.ScalFunctionOfScal)
   spis.Util.List.GlobalMaxwellianDataProvider (implements spis.Util.List.MaxwellianDataProvider)
   spis.Surf.SurfField.GlobalTempCurrentScalerDeprecated (implements spis.Surf.SurfField.CurrentScaler)
   spis.Util.Func.GradCosMultiD (implements spis.Util.Func.VectFunctionOfVect)
   spis.Util.Func.GradPowerLaw (implements spis.Util.Func.VectFunctionOfVect)
   spis.Top.Grid.Grid (implements spis.Util.Monitor.NumericsMonitorable)
      spis.Top.Grid.InteractGrid
   spis.Util.Func.Identity (implements spis.Util.Func.ScalFunctionOfScal)
   spis.Util.io.InputReader
   spis.Util.Instrument.InstrumentsCatalogue
   spis.Util.OcTreeMesh.IntComparator (implements java.util.Comparator<T>)
   spis.Surf.SurfInteract.InteractModel
      spis.Surf.SurfInteract.MaterialModel
         spis.Surf.SurfInteract.BasicMaterialModel
            spis.Surf.SurfInteract.ErosionMaterialModel
   spis.Surf.SurfInteract.Interactor (implements spis.Util.Monitor.Monitorable)
      spis.Surf.SurfInteract.BasicInducedConductInteractor
      spis.Surf.SurfInteract.CathodeSpot
      spis.Surf.SurfInteract.CathodeSpotElectron
      spis.Surf.SurfInteract.CathodeSpotIon
      spis.Surf.SurfInteract.ErosionInteractor
      spis.Surf.SurfInteract.GenericDFInteractor
         spis.Surf.SurfInteract.GenericPhotoEmInteractor
         spis.Surf.SurfInteract.GenericSEEPInteractor
      spis.Surf.SurfInteract.MaxwellianInteractor
         spis.Surf.SurfInteract.BasicPhotoEmInteractor
         spis.Surf.SurfInteract.BasicSEEPInteractor
         spis.Surf.SurfInteract.MaxwellianInteractorWithRecollection
            spis.Surf.SurfInteract.ImprovedPhotoEmInteractor
         spis.Surf.SurfInteract.RecollPhotoEmInteractor
      spis.Surf.SurfInteract.MultipleInteractor
         spis.Surf.SurfInteract.BasicSEEEInteractor
         spis.Surf.SurfInteract.GenericSEEEInteractor
      spis.Surf.SurfInteract.MultipleMaxwellianInteractor
      spis.Surf.SurfInteract.PICInteractor
      spis.Surf.SurfInteract.ReflectionInteractor
      spis.Surf.SurfInteract.SolarArrayInteractor
      spis.Surf.SurfInteract.SolarArrayInteractor2
      spis.Surf.SurfInteract.Source
   spis.Surf.SurfInteract.InteractParamSet (implements java.io.Serializable)
      spis.Surf.SurfInteract.ErosionParamSet
      spis.Surf.SurfInteract.GenericParamSet (implements java.io.Serializable)
      spis.Surf.SurfInteract.NascapParamSet
   spis.Util.io.Introspection
   spis.Util.io.IOUtilities
   spis.Util.Func.IsDifferent (implements spis.Util.Func.TestOfInt)
   spis.Util.Func.Kappa1_1D (implements spis.Util.Func.ScalFunctionOfScal)
   spis.Util.Func.Kappa1_3D (implements spis.Util.Func.ScalFunctionOfScal)
   spis.Solver.Util.LeastSquare
   spis.Util.List.LocalMaxwellianDataProvider (implements spis.Util.List.MaxwellianDataProvider)
   spis.Util.Func.LowerGammaIncFunction (implements spis.Util.Func.InvertableScalFunctionOfScal, spis.Util.Func.ScalFunctionOfScal)
   spis.Util.Func.MaskedIsDifferent (implements spis.Util.Func.TestOfInt)
   spis.Util.Matrix.Matrix (implements java.io.Serializable)
      spis.Util.Matrix.DiagMatrix (implements java.io.Serializable)
      spis.Util.Matrix.DirMatrix (implements java.io.Serializable)
         spis.Util.Matrix.DirMatrixDouble (implements java.io.Serializable)
      spis.Util.Matrix.Rank1Matrix
      spis.Util.Matrix.SparseMatrix (implements java.io.Serializable)
      spis.Util.Matrix.SparseMatrixDeprecated (implements java.io.Serializable)
   spis.Vol.BC.MatterBC
      spis.Vol.BC.ScaledSurfDistribMatterBC
      spis.Vol.BC.SurfDistribMatterBC
         spis.Vol.BC.OneSurfDistribTestableMatterBC (implements spis.Vol.BC.TestableMatterBC)
      spis.Vol.BC.VoltageDependentMBC
         spis.Vol.BC.VoltageGenerator
            spis.Vol.BC.CapacitiveVoltageGenerator
   spis.Util.Func.Maxwellian1D (implements spis.Util.Func.ScalFunctionOfScal)
   spis.Util.Monitor.Monitor
   spis.Surf.SurfField.MultipleCurrentScaler (implements spis.Surf.SurfField.CurrentScaler)
   spis.Surf.SurfField.MultipleScalSurfField
   spis.Circ.Circ.MultipleSurfDIDV (implements spis.Circ.Circ.SurfDIDV)
   spis.Solver.Matter.MultiZone
      spis.Solver.Matter.BoltzmannPICMultiZone
   spis.Util.List.NoStatisticsOptimizer (implements spis.Util.List.StatisticsOptimizerI)
   spis.Circ.Circ.NullSurfDIDV (implements spis.Circ.Circ.SurfDIDV)
   spis.Util.Monitor.NumericsMonitor
   spis.Top.Top.NumTopFromUI (implements spis.Top.Top.UIInvokable)
   spis.Util.OcTree.OcTree
      spis.Util.OcTree.ScalOcTree
         spis.Util.OcTree.TrackedParticlesOcTree
   spis.Util.OcTreeMesh.OcTreeLine
   spis.Util.OcTreeMesh.OcTreeMesh
   spis.Util.OcTreeMesh.OcTreePoint
   spis.Util.OcTreeMesh.OcTreeSurface
   spis.Util.OcTreeMesh.OcTreeVolume
      spis.Util.OcTreeMesh.OcTreeSlab
   spis.Solver.Matter.OldComplexPusher (implements spis.Solver.Matter.ParticlePusher)
   spis.Util.List.OneDSampler (implements spis.Util.List.OneDSamplable)
      spis.Util.List.MaxwellianEnergySampler
   spis.Util.Func.Or (implements spis.Util.Func.TestOfInt)
   spis.Top.Default.Parameter (implements java.io.Serializable)
      spis.Top.Default.GlobalParameter (implements java.io.Serializable)
      spis.Top.Default.InstrumentParameter
      spis.Top.Default.LocalParameter (implements java.io.Serializable)
      spis.Top.Default.MaterialProperty (implements java.io.Serializable)
   spis.Surf.SurfInteract.ParamSetDeprecated
   spis.Surf.SurfInteract.ParamSetExtractor
   spis.Surf.SurfInteract.ParamSetExtractorDeprecated
   spis.Util.Part.Part
   spis.Util.Monitor.ParticleMeasurementResult (implements java.io.Serializable)
   spis.Util.Part.PartTable
   spis.Util.Phys.Phys
   spis.Top.Plasma.Plasma (implements spis.Util.Monitor.Monitorable, spis.Util.Monitor.NumericsMonitorable)
      spis.Top.Plasma.MeshedPlasma
         spis.Top.Plasma.MmfPlasma
            spis.Top.Plasma.FlexibleMFPlasmaDeprecated
            spis.Top.Plasma.Hybrid1MmfPlasma
   spis.Util.List.PointList (implements java.io.Serializable)
      spis.Util.List.PartList
         spis.Util.List.NotTrackedPartList
         spis.Util.List.RichPartList
            spis.Util.List.FlexiblePartList
   spis.Util.Instrument.PointPlasmaSensor (implements org.spis.instruments.Instrument<MeasureType,ParameterType>)
      spis.Util.Instrument.SphericalPlasmaSensor
   spis.Solver.Poisson.PoissonInit
   spis.Solver.Poisson.PoissonSolve
   spis.Solver.Poisson.PoissonSolver
      spis.Solver.Poisson.PotPoissonSolver
         spis.Solver.Poisson.ConjGrad3DUnstructPoissonSolver
   spis.Util.Func.Power (implements spis.Util.Func.ScalFunctionOfScal)
   spis.Util.Func.PowerLaw (implements spis.Util.Func.ScalFunctionOfVect)
   spis.Util.Func.ProductOfTwoScalFunctionOfScal (implements spis.Util.Func.ScalFunctionOfScal)
   spis.Circ.Circ.RCCabsCirc (implements spis.Circ.Circ.CircIf)
      spis.Circ.Circ.RLCCirc (implements spis.Circ.Circ.ImplicitCircIf)
   java.io.Reader (implements java.io.Closeable, java.lang.Readable)
      java.io.InputStreamReader
         java.io.FileReader
            spis.Util.io.SpisFileReader
      java.io.StringReader
         spis.Util.io.SpisStringReader
   spis.Util.Func.ReciprocalFunction (implements spis.Util.Func.ScalFunctionOfScal)
   spis.Util.List.Sampling
   spis.Top.SC.SC (implements spis.Util.Monitor.NumericsMonitorable)
      spis.Top.SC.InteractSC
         spis.Top.SC.EquipotSC
         spis.Top.SC.RCCabsSC
            spis.Top.SC.RLCSC
   spis.Util.Func.ScalFuncOfScalFromTestOfInt (implements spis.Util.Func.ScalFunctionOfScal)
   spis.Util.Func.ScalFunctionOfScalFromFunctionOf4 (implements spis.Util.Func.ScalFunctionOfScal)
      spis.Util.Func.InvertibleScalFunctionOfScalFromFunctionOf4 (implements spis.Util.Func.InvertableScalFunctionOfScal)
   spis.Util.Func.ScalFunctionOfScalFromFunctionOf5 (implements spis.Util.Func.ScalFunctionOfScal)
   spis.Top.Top.Scenario (implements spis.Top.Top.UIInvokable)
```

```
  spis.Top.Top.PotentialSweep
spis.Util.Func.SeparableFunctionOf3Scal (implements spis.Util.Func.ScalFunctionOf3Scal)
spis.Util.io.Serialisation
spis.Util.Func.Set (implements spis.Util.Func.ModifInt)
spis.Util.Func.ShiftSet (implements spis.Util.Func.CombineInt)
spis.Top.Simulation.Simulation (implements spis.Util.Monitor.NumericsMonitorable)
  spis.Top.Simulation.PlasmaScSimulation
    spis.Top.Simulation.SimulationFromUIParams (implements spis.Top.Top.UIInvokable)
spis.Top.Transition.SimulationParamUpdater
  spis.Top.Transition.Finalization
  spis.Top.Transition.RCCabsSCUpdater
  spis.Top.Transition.SheathOrPresheathPoissonBCUpdater
  spis.Top.Transition.SourceFluxUpdater
  spis.Top.Transition.SunFluxIntensityUpdater
  spis.Top.Transition.SunFluxUpdater
  spis.Top.Transition.TimeStepForBETs
  spis.Top.Transition.VcrossBfieldUpdater
spis.Solver.Util.SolverUtil
spis.Top.Default.SpisDefaultMaterials
spis.Top.Default.SpisDefaultPartTypes
spis.Top.Default.SpisDefaultSampling
spis.Util.io.SpisIO
spis.Top.Top.SpisTopMenu
spis.Util.Func.Sqrt (implements spis.Util.Func.ScalFunctionOfScal)
spis.Util.io.StandardInputReader
spis.Util.List.StatisticsCollector (implements spis.Util.List.StatisticsCollectorI)
spis.Util.List.StatisticsOptimizer (implements spis.Util.List.StatisticsOptimizerI)
spis.Circ.Circ.SurfDIDVFromMatrices (implements spis.Circ.Circ.SurfDIDV)
spis.Surf.SurfDistrib.SurfDistrib (implements spis.Surf.SurfDistrib.SurfDistribTag)
  spis.Surf.SurfDistrib.MultipleSurfDistrib
  spis.Surf.SurfDistrib.NonPICSurfDistrib
    spis.Surf.SurfDistrib.FluidSurfDistrib
      spis.Surf.SurfDistrib.AisepsCathodeSurfDistrib
      spis.Surf.SurfDistrib.AisepsThrusterSurfDistrib
      spis.Surf.SurfDistrib.AxisymTabulatedVelocitySurfDistrib
      spis.Surf.SurfDistrib.LocalGenericSurfDistrib
      spis.Surf.SurfDistrib.LocalMaxwellSurfDistrib
        spis.Surf.SurfDistrib.FlexibleSurfDistrib
        spis.Surf.SurfDistrib.FowlerNordheimSurfDistrib
        spis.Surf.SurfDistrib.LocalMaxwellSurfDistribWithMachNotOperational
        spis.Surf.SurfDistrib.MaxwellianThruster
        spis.Surf.SurfDistrib.RecollMaxwellSurfDistrib
      spis.Surf.SurfDistrib.TwoAxesTabulatedVelocitySurfDistrib
      spis.Surf.SurfDistrib.UniformVelocitySurfDistrib
    spis.Surf.SurfDistrib.GlobalSurfDistrib
      spis.Surf.SurfDistrib.GenericSurfDistrib (implements spis.Surf.SurfDistrib.TestableSurfDistrib)
      spis.Surf.SurfDistrib.GlobalMaxwellSurfDistrib
        spis.Surf.SurfDistrib.GlobalMaxwellBoltzmannSurfDistrib (implements spis.Surf.SurfDistrib.TestableSurfDistrib)
        spis.Surf.SurfDistrib.GlobalMaxwellBoltzmannSurfDistrib2 (implements spis.Surf.SurfDistrib.TestableSurfDistrib)
      spis.Surf.SurfDistrib.GlobalMaxwellSurfDistrib2
  spis.Surf.SurfDistrib.PICSurfDistrib
spis.Surf.SurfField.SurfField (implements java.io.Serializable)
  spis.Surf.SurfField.ScalSurfField (implements java.io.Serializable)
    spis.Surf.SurfField.ScalableCurrent
  spis.Surf.SurfField.VectSurfField (implements java.io.Serializable)
spis.Surf.SurfInteract.SurfInteractDeprecated
spis.Surf.SurfMesh.SurfMesh (implements java.io.Serializable)
    spis.Surf.SurfMesh.UnstructSurfMesh (implements java.io.Serializable)
        spis.Surf.SurfMesh.ThreeDUnstructSurfMesh (implements java.io.Serializable)
spis.Util.List.SurfSampler (implements spis.Util.List.SurfVeloSampler)
    spis.Util.List.AISEPSLocalSurfSampler
    spis.Util.List.GenericSurfSampler
    spis.Util.List.GlobalMaxwellSurfSampler
        spis.Util.List.GlobalMaxwellSurfSampler1
            spis.Util.List.GlobalMaxwellSurfSampler2
                spis.Util.List.GlobalMaxwellSurfSamplerMuscatSpisWake
        spis.Util.List.GlobalMaxwellSurfSampler1withPot
    spis.Util.List.LocalSurfSampler
        spis.Util.List.AxisymTabulatedSampler
        spis.Util.List.LocalMaxwellSurfSampler
            spis.Util.List.LocalMaxwellSurfSampler1
                spis.Util.List.LocalTrunckatedMaxwellSurfSampler
                spis.Util.List.RecollLocalMaxwellSurfSampler
            spis.Util.List.MaxwellianThrusterSampler
        spis.Util.List.Spherical3dTabulatedSampler
        spis.Util.List.TwoAxesTabulatedSampler
        spis.Util.List.UniformVeloSampler
spis.Util.Table.Table (implements java.io.Serializable)
    spis.Util.Table.ScalTable (implements java.io.Serializable)
        spis.Util.Table.DoubleScalTable
        spis.Util.Table.FloatScalTable (implements java.io.Serializable)
            spis.Util.Phys.DimVect (implements java.io.Serializable)
        spis.Util.Table.IntScalTable (implements java.io.Serializable)
    spis.Util.Table.VectTable (implements java.io.Serializable)
        spis.Util.Table.SpaceVectTable (implements java.io.Serializable)
        spis.Util.Table.VeloVectTable (implements java.io.Serializable)
spis.Util.Func.TabulatedScalFunctionOf3Scal (implements spis.Util.Func.ScalFunctionOf3Scal)
spis.Util.Func.TabulatedScalFunctionOfScal (implements spis.Util.Func.ScalFunctionOfScal)
spis.Util.Monitor.TaskComputingDuration
spis.Solver.Matter.TaskOfParticlePusher (implements java.util.concurrent.Callable<V>)
spis.Top.Simulation.testScenario
java.lang.Thread (implements java.lang.Runnable)
    spis.Top.Plasma.PlasmaPopMoveInThread
    spis.Solver.Matter.ThreadOfParticlePusher
spis.Vol.Geom.ThreeDCartesianGeom (implements spis.Vol.Geom.Geom, java.io.Serializable)
spis.Util.List.ThreeDSphericalSampler
    spis.Util.List.TonduErodedProductSampler
java.lang.Throwable (implements java.io.Serializable)
    java.lang.Exception
        java.lang.RuntimeException
            spis.Util.Exception.SpisRuntimeException
                spis.Util.Exception.SpisParameterNotFoundException
spis.Util.Monitor.TimeWatch
spis.Util.Instrument.TopInstrumentFactory (implements org.spis.instruments.InstrumentFactory)
spis.Util.OcTree.TrackedParticle
spis.Util.Monitor.Trajectory (implements java.io.Serializable)
spis.Top.Transition.Transition
    spis.Top.Transition.BasicEclipseExit
    spis.Top.Transition.ConductivityEvolution
    spis.Top.Transition.LangmuirProbeTransition
```

```
  spis.Top.Transition.SpinningSpacecraft
  spis.Top.Transition.TransientArtificialSources
spis.Top.Transition.TransitionObserver
spis.Vol.Geom.TwoThreeDAxisymGeom (implements spis.Vol.Geom.Geom, java.io.Serializable)
spis.Util.Phys.Unit (implements java.io.Serializable)
  spis.Util.Phys.Unit1 (implements java.io.Serializable)
spis.Util.Func.UnitUtil
spis.Util.Vect.Vect
spis.Vol.VolDistrib.VolDistrib (implements spis.Util.Monitor.NumericsMonitorable)
  spis.Vol.VolDistrib.BoundaryFlux
  spis.Vol.VolDistrib.VolDistribWithIO
    spis.Vol.VolDistrib.CompositeVolDistrib
      spis.Vol.VolDistrib.BacktrackingBoltzmannCompositeVolDistrib
      spis.Vol.VolDistrib.BacktrackingPICCompositeVolDistrib
    spis.Vol.VolDistrib.HybridMZVolDistrib
      spis.Vol.VolDistrib.NoSinkHMZVD
    spis.Vol.VolDistrib.NonPICVolDistrib
      spis.Vol.VolDistrib.AnalyticVolDistrib
        spis.Vol.VolDistrib.AdiabaticMBVolDistrib
        spis.Vol.VolDistrib.GlobalMaxwellBoltzmannVolDistrib
          spis.Vol.VolDistrib.GlobalMaxwellBoltzmannVolDistribWithNeutrality
          spis.Vol.VolDistrib.UnlimitedGlobalMaxwellBoltzmannVolDistrib
            spis.Vol.VolDistrib.TrunckatedGlobalMaxwellBoltzmannVolDistrib
      spis.Vol.VolDistrib.LocalMaxwellVolDistrib
    spis.Vol.VolDistrib.PICVolDistrib
      spis.Vol.VolDistrib.BackTrackingVolDistrib
      spis.Vol.VolDistrib.PICVolDistrib2
      spis.Vol.VolDistrib.PICVolDistribNoAcc
      spis.Vol.VolDistrib.SmartPICVolDistrib
    spis.Vol.VolDistrib.PICVolDistribOld
spis.Vol.VolField.VolField (implements java.io.Serializable)
  spis.Vol.VolField.ScalVolField (implements java.io.Serializable)
  spis.Vol.VolField.VectVolField (implements java.io.Serializable)
    spis.Vol.VolField.DirVectVolField (implements java.io.Serializable)
    spis.Vol.VolField.PotVectVolField (implements java.io.Serializable)
spis.Vol.VolInteract.VolInteractor (implements spis.Util.Monitor.Monitorable)
  spis.Vol.VolInteract.DustChargingInteractor
  spis.Vol.VolInteract.MCCInteractor
    spis.Vol.VolInteract.CEXInteractor
    spis.Vol.VolInteract.CEXInteractor3
    spis.Vol.VolInteract.ConstantIonizationInteractor
    spis.Vol.VolInteract.ElasticCollisions
    spis.Vol.VolInteract.PhotoIonization
spis.Vol.VolMesh.VolMesh (implements spis.Util.Monitor.NumericsMonitorable, java.io.Serializable)
  spis.Vol.VolMesh.UnstructVolMesh
    spis.Vol.VolMesh.ThreeDUnstructVolMesh
spis.Util.List.VolSampler
  spis.Util.List.GenericVolSampler
  spis.Util.List.GlobalMaxwellVolSampler
    spis.Util.List.GlobalMaxwellVolSampler1
  spis.Util.List.MaxwellVolSampler2
spis.Util.Monitor.XyData (implements java.io.Serializable)
  spis.Util.Monitor.DocumentedXyData
spis.Util.Monitor.XyzData (implements java.io.Serializable)
spis.Util.Monitor.XyzDataRect (implements java.io.Serializable)
```

2.3.3. SPIS-UI

In parallel of the evolution of the simulation kernel (SPIS-NUM), we remind that since the version 5.0, initially developed in the frame of SPIS-GEO/MEO, the SPIS-UI framework has been fully redeveloped and is now based on the Artenum's Keridwen Integrated Modelling Environment (IME). Around a central kernel (controler), Keridwen is designed as a modular framework where each functional block is dynamically integrated under the form of normalised OSGI bundles. The list of loaded bundles and the deployment order of these ones are defined through XML based configuration files without any recompilation.

Without modification of the core software, such approach allows to easily adapt the whole framework to the specificities of each declination (e.g. SPIS-GEO, SPIS-SCI...) by simple setting of the configuration files and selection of relevant bundles.

SPIS modules gather the developments and functionalities specifically related to SPIS activities. In order to reduce and mutualised as possible the maintenance effort, as much as possible generic components, like mesh loaders or the Groups Editor, are based on pre-existing and already validated external components in Keridwen or other dependencies.

Technically, Keridwen is divided into two sub-sets:

- Keridwen core, which corresponds to the core library (e.g. interfaces) and low levels components (e.g controller);
- Keridwen tools, which gather generics tools like CAD tool, mesh Manager, Groups manager...;

By transitivity, SPIS integrates also other external components as software components (e.g. VTK, JFreeChart, logging system, etc...) or as external components called through a system call (e.g. Gmsh).

Figure 11, here after, illustrates the collaboration/inter-dependency tree between the various OSGi modules. This scheme shows precisely OSGi modules being part of SPIS and other software. External components called by transitivity, like Gmsh, JFreeChart or VTk, are not represented here. Most of them are not called directly from SPIS but through external abstracting components, like Keridwen, Cassandra or Penelope.

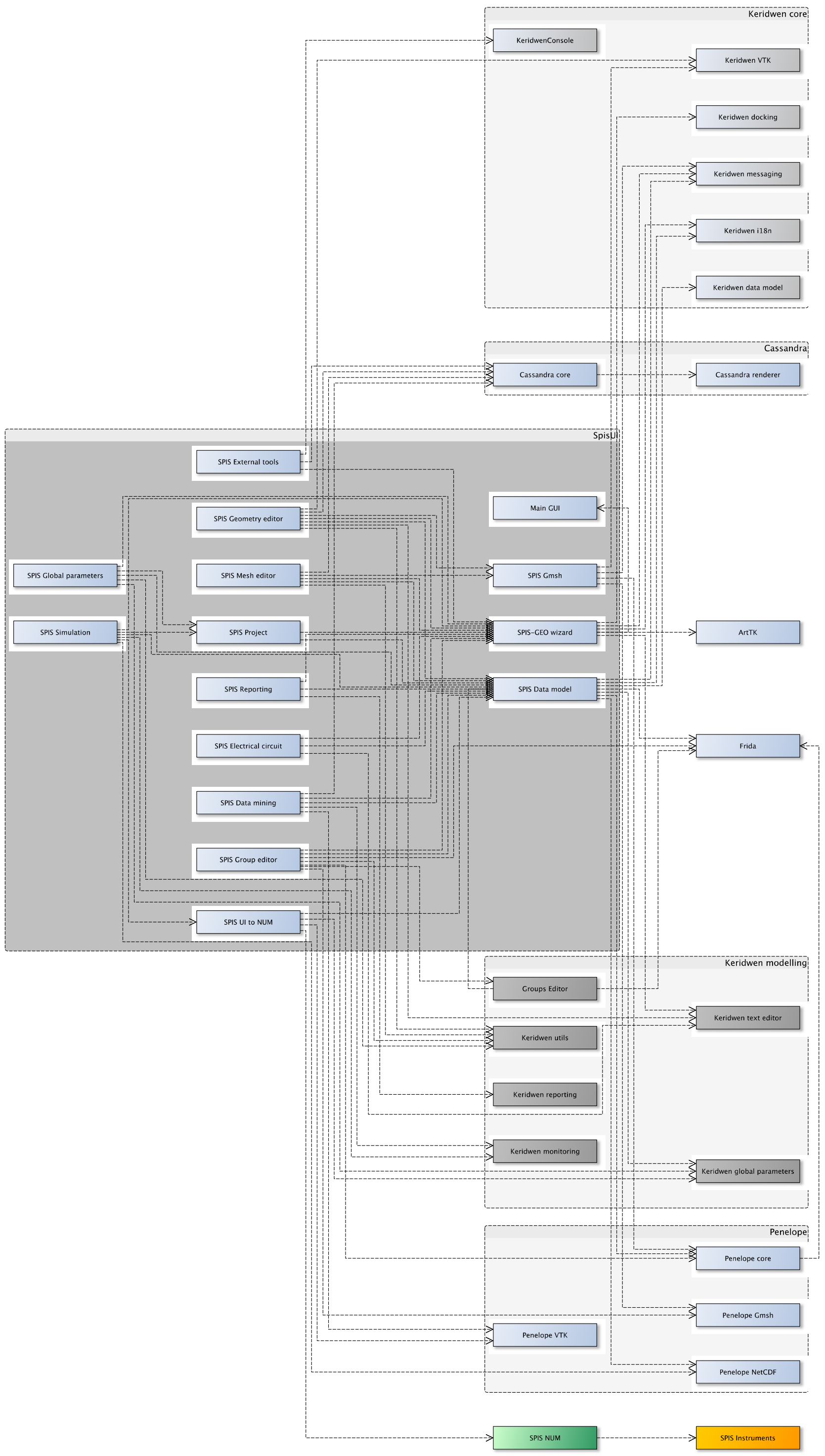


Figure 11: Overview of the whole SPIS architecture and OSGi modules decomposition. Only SPIS and directly related external modules are illustrated here. Transitive dependencies, like VTK, are not shown in this diagram.

With respect to the previous versions of SPIS (4.3 and earlier), this choice of the OSGI industrial and reference standard should offer an improved stability, reduced maintenance cost (by mutualisation of common components) and a better potential interoperability with other modelling tools used at TEC-EES.

On this basis, a specific central GUI for SPIS and dedicated modules (e.g. bundle of integration of SPIS-NUM integration and simulation driver) has been developed in the frame of SPIS-GEO. Such SPIS related modules include the following bundle (the project in which one the main part of the development have been done is given in parenthesis):

- **org-spis-ui-data-mining:** Specific tailored declination of the Keridwen data-miner for SPIS (SPIS-GEO);
- **org-spis-ui-electrical-circuit:** Editor for the definition of the internal circuit;
- **org-spis-ui-geo-wizard:** Tailored adaptation of the Keridwen wizard manager for SPIS-GEO. The model process of SPIS-SCI is based on the same wizard manager with a dedicated scenario (XML configuration file);
- **org-spis-ui-geometry-editor:** Tailored integration/adaptation of the Keridwen Geomertry Manager. This component has been deeply updated to take into account the specificities of SPIS-SCI (handling of several CAD files for instruments, STG, etc...);
- **org-spis-ui-global-parameters:** Tailored declination of the Keridwen Global Parameters Editor for SPIS, including the handling of pre-set parameter lists (i.e. geo standard and worst cases) and extension to extra parameters required by SPIS-SCI models.
- **org-spis-ui-gmsh:** Tailored declination of the Keridwen Mesh Editor for SPIS. This module has been deeply refactored in the frame of SPIS-SCI to handle extra elements like instruments and STG.
- **org-spis-ui-group-editor:** Integration of the Keridwen Groups Editor, with the development of a set of dedicated extra filters and pre-processors specific to SPIS (dynamic loading):
    - **SpisDefaultTypesFilters:** Dedicated property types to group types pre-presetting filter, to simplify the property to groups attribution (development SPIS-GEO, update SPIS-SCI to handle STG and instruments);
    - **MeshGroupSplitter:** Dedicated group operator to perform the mesh splitting needed by 2D thin elements (development SPIS-GEO, update SPIS-SCI);
- **org-spis-ui-instruments:** Integration bundle of the SPIS related instruments (SPIS-SCI development);
- **org-spis-ui-model:** Tailored declination of the central data model for SPIS (SPIS-GEO development);
- **org-spis-ui-project:** Handling of the SPIS 5.0 persistence scheme (SPIS-GEO development);
- **org-spis-ui-project-converter:** SPIS legacy (4.3 and earlier) to SPIS 5.0 and higher project conversion bundle (SPIS-GEO development, update SPIS-SCI);
- **org-spis-ui-reporting:** Specific setting and integration of the Keridwen auto-reporting module for SPIS (SPIS-GEO development);
- **org-spis-ui-simulation:** Dedicated SPIS-NUM integration module and simulation control. This module integrates the real time monitoring functions (through the SPIS instruments) and the pause/start function of the simulation look. This module also performs the instantiation of requested SPIS intruments. This module has been specifically developed in the frame of SPIS-GEO, but has been deeply refactored in the frame of SPIS-SCI. See the specific TN regarding its detailed implementation in the SPIS documentation.
- **org-spis-ui-tools:** Various tools for SPIS.
- **org-spis-ui-ui2num:** This module is the specific module of interfacing the generic data structures of SPIS-UI to the specific one of SPIS-NUM. This modules performs, for instance, the volume mesh conversion, the generation of the specific surface meshes (i.e. spacecraft surface, external boudnaries, instruments meshes, STG meshes), the mapping of all relevant pre-processing data fields (i.e.

DataFields to LocalParam conversion), the conversion of all needed properties, like materials, into SPIS-NUM GenericParamSet and the instantiation of the simulation object (SPIS-NUM). This module has been specifically developed in the frame of SPIS-GEO, but has been deeply refactored in the frame of SPIS-SCI to take into account of new evolutions of the API of control of SPIS-NUM and the support of extra functionalities (extended materials and properties characterisation, support of instruments, support of STG, improvement and extension of the data miner). See the specific TN regarding its detailed implementation in the SPIS documentation.

We remind that the whole compilation process of SPIS is based on Apache Maven. This tool manages by its own the needed dependencies during the compilation, through normalised Maven artefacts. All of these ones are stored on online Maven repositories. Artefacts corresponding to Artenum's products are stored on the public Artenum's Maven repository (see maven.artenum.com).

We outline that if these various OSGI modules are dynamically loaded at the starting of the framework, they might be inter-dependant. As a consequence, if the configuration scripts of various SPIS branches (SPIS-GEO, SPIS-SCI) should be updated in future evolutions, the dependence tree should carefully check before.

2.3.4. Instruments

In the frame of SPIS-SCI, an extra library, called Spis Instruments, has been developed to provide a normalised interface and API for instruments. This library is developed and maintained in a separated package from SPIS-NUM and SPIS-UI to avoid direct cross dependency outside the existing bundles of interfacing.

Spis Instrument has fully been developed in the frame of SPIS-SCI, however, and in a wish of normalisation and simplification of the maintenance, most of the monitoring tools of SPIS-GEO/MEO have been rewritten on the basis of the normalised Spis Instrument interface.

The Spis Instrument package is detailed later on in the present document. The global interfacing with other components is described in paragraph 3.2.1.

2.4. External components

The Interface with external components is described in the "Interface Control Document (ICD) and Software Life Cycle (SLC)" document.

3. DETAILED DESIGN

This section presents the detailed design of SPIS-SCIENCE developments. The matrix of compliance between the SRs and the developed routines is given in 3.1. For the sake of clarity, next sections follow the same order as in the SRD [AD4].

3.1. Matrix between SR and Developments

The matrix between each (group of) SR(s) [AD4] and the sections describing the software developments of SPIS-SCI is presented below.

SR or group of SR	Title	Design described in section #
SR-PD-001	Particle Detector java OO interface	3.2.3
SR-PD-002	Particle Detector User Interface	3.2.3
SR-PD-003	Virtual particle detector meshing	3.2.5
SR-PD-004	Advanced particle pusher	3.2.3.3 and 3.2.5.2
SR-PD-006	Particle counting 2: Test Particle method	3.2.3.3 and 3.2.5.2
SR-PD-010	Advanced TP method	3.2.3.3 and 3.2.5.2
SR-PD-011	Basic particle detector outputs	3.2.3.4
SR-PD-012	Post-processed distributions functions	3.2.3.4 and 3.2.3.5
SR-PD-013	Particle detector transfer function	3.2.3.4 and 3.2.3.5
SR-PD-014	Monitoring of tracked trajectories	3.2.3.4 and 3.2.3.5
SR-VPS-001	java OO interface for Virtual Plasma Sensor	3.2.6
SR-VPS-002	Virtual Plasma Sensor User Interface	3.2.6
SR-VPS-003	Point plasma sensors	3.2.6
SR-VPS-004	Spherical plasma sensors	3.2.6
SR-STG-001	CAD modelling of STG	3.3.1
SR-STG-002	STG potential	3.3
SR-STG-003	STG collected current	3.3
SR-STG-004	STG emitted current	3.3
SR-ESC-001	Distribution functions for ambient plasma	3.4
SR-ESC-002	Upgraded matter boundary conditions	3.9.2.1
SR-ESC-003	Upgraded electric field boundary conditions	3.9.2.2
SR-ESC-004	Upgraded particle transport model	3.8.1

SR or group of SR	Title	Design described in section #
SR-PS-001	Double maxwellian photoelectron distribution	3.4.3 and 3.5
SR-PS-002	Isotropic photoelectron distribution	3.4.2 and 3.5
SR-PS-006	Self-shadowing	3.5.3
SR-PS-008	Isotropic SEEE distribution	3.5.2.2
SR-PS-010	SEEE yield user definition	3.5.2.1
SR-PS-011	Photo ionisation of neutrals	not described, see SUM
SR-FGS-001	Thin planes	3.6
SR-FGS-002	Solar arrays plasma interaction	3.5.4
SR-FGS-003	Motional electric field E = v × B.	3.8.2
SR-FGS-004	Thin wires- thin booms	3.6
SR-SP-001	Advanced material properties	3.5.1
SR-SP-002	Improvement of UI/NUM interface	2.3 and 3.2.2.3
SR-SP-003	Parameter updater class	3.7
SR-SP-004	Spinning Spacecraft Updater	3.7
SR-SP-005	Plasma source updater	3.7
SR-SP-006	Bias voltage updater	3.2.4 and 3.7
SR-SP-008	Scenario/Transition overlayer	3.7
SR-SP-009	Scenario for spinning spacecraft	3.7
SR-SP-010	Scenario for plasma sources	3.7
SR-SP-011	Scenario for bias voltage	3.2.4 and 3.7
SR-SP-014	Stopping and restarting a simulation	3.10.2
SR-SP-015	S/C surface Potential vs. time monitoring	not described, see Cassandra documentation.
SR-PE-002	Exact varying number of super particles	3.9.1
SR-PE-004	Fast run speed	3.9.3, 3.9.4, 3.9.5

3.2. Instruments

3.2.1. Objectives

The objective of the instruments is to provide the user with specific information on the simulation outputs. The results provided depend on the sub-type of instruments.

In order to simplify and normalize the access to scientific data, observations (i.e. data extraction) are done through a set of classes of instruments, each of them implementing a normalized interface called Instrument. The Instrument interface offers a normalized API to extract scientific data and information from the simulation kernel and return them to the modeling framework under a standardized form. Each concrete implementation of the Instrument interface should implement it.

As real scientific instruments and/or detectors, Instruments can be executed in two modes:

- On user demand, from the modeling framework, for a punctual and specific observation;
- On the basis of regular observations using a sample frequency defined at the framework level;

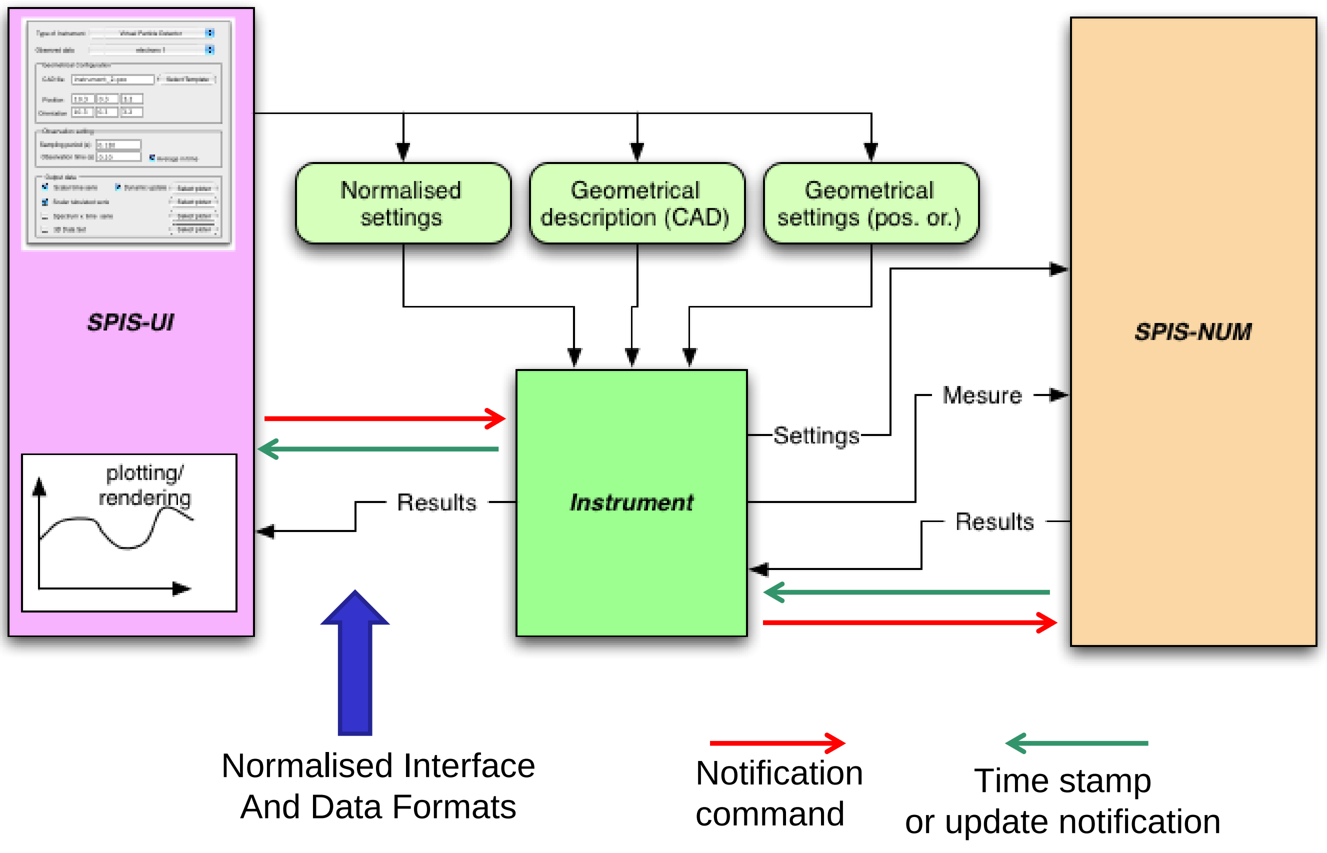
Outputs from Instruments can be:

- DataFields of local data deployed on the grid (e.g. maps of potential);
- Times series, including spectrograms;
- Tabulated data and histograms.

Settings are previously defined through the pre-processing information like , global parameters, geometrical description and settings.

A notification/listener system between the Instrument and the simulation kernel has been implemented.

The Instrument access directly to the SPIS–NUM objects and is itself accessible from SPIS-UI. Next figure summarizes this approach and provides a simplified collaboration diagram.



UNCLASSIFIED

3.2.2. Global design

3.2.2.1. Implementation of the Instrument Interface

The UML diagram in Figure 12 gives a first proposition of API of the Instrument interface and its associations with the SimulationFromUIParams class. It follows the Observer/Observable pattern. The observable is the simulation (the observed parameter being the current physical time of the simulation). The observers are the realizations of the interface Instrument that are added to the simulation.

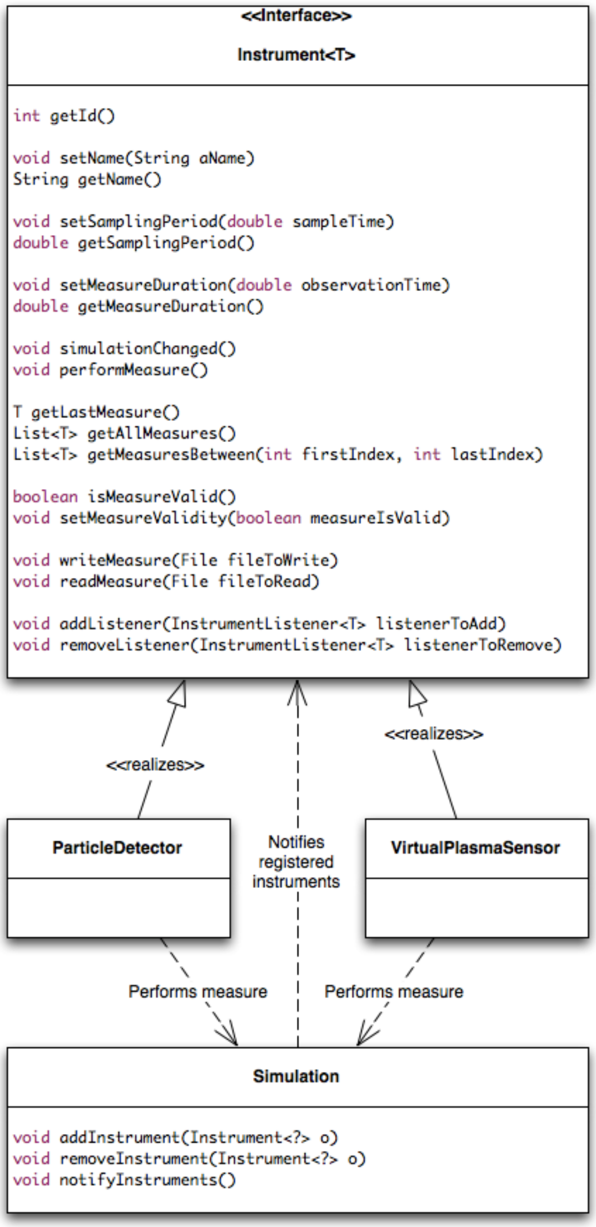

<<Interface>>
Instrument<T>
int getId()
void setName(String aName)
String getName()
void setSamplingPeriod(double sampleTime)
double getSamplingPeriod()
void setMeasureDuration(double observationTime)
double getMeasureDuration()
void simulationChanged()
void performMeasure()
T getLastMeasure()
List<T> getAllMeasures()
List<T> getMeasuresBetween(int firstIndex, int lastIndex)
boolean isMeasureValid()
void setMeasureValidity(boolean measureIsValid)
void writeMeasure(File fileToWrite)
void readMeasure(File fileToRead)
void addListener(InstrumentListener<T> listenerToAdd)
void removeListener(InstrumentListener<T> listenerToRemove)
<<realizes>>
<<realizes>>
ParticleDetector
Notifies registered instruments
VirtualPlasmaSensor
Performs measure
Performs measure
Simulation
void addInstrument(Instrument<?> o)
void removeInstrument(Instrument<?> o)
void notifyInstruments()

Figure 12 - Detailed API of the Instrument interface and example of the inheritance scheme for various implementations

Each instrument should implement this interface. The Instrument is identified through its Id. A Name completes this identification for the user. Accessors have been defined for both. The measure is started on request through the performMeasure method or every Tsample duration. Tsample corresponds to the sampling period. Because, some types of measurements require an observation duration, a Tobserve, is defined. For both accessors have been implemented. The performMeasure method can be called by both the SPIS-UI and the SPIS-NUM layers. By this all use modes can be done.

The methods isMeasureValid() et setMeasureValidity() handles interactions with the SPIS-NUM kernel and the user through GUI. It is used to determine if the performMeasure method can be considered as successful or if a second run with different parameters is necessary. This method is concretely realized by classes of instruments that really offer the possibility for the user to interact with. It is for example the case of particle detectors (as e.g. the energy range of the test particle method can be interactively refined by the user).

The getMeasure method returns the result of the measure to SPIS-UI under the form of a normalised form. A MethodType field (String) defined in static for each implementation allows selecting the type of measure returned if the current instrument can generate several types of outputs.

In order to notify an external components that the measure is valid (for instance), listener have been added through the addListener method. This allows the UI level to be notified when a measure is ready at the Instrument level itself or from the SPIS-NUM level.

3.2.2.2. Integrated Data structure and interface

Each implementation of the Instrument interface should type its output data (T parameter in the generic characterization). In this sense, the instruments can be returned only on output data, but this one can be a complex object structure, here after caller MeasuredData, gathering the different outputs. The selection of the relevant physical output is then done on the MeasuredData object itself.

MeasureData include meta-data for a better tracking of the measured data (such as time).

If, needed, the same approach can be done on the NUM layer to define MeasuredData objects independent on external libraries.

3.2.2.3. Collaborative process UI/Instruments/NUM

In order to keep instruments independent on tailored low-level libraries, like SPIS-NUM, it is proposed that the Instrument interface and its implementations be packaged in an independent package. Such approach allows using them independently in each level (i.e. SPIS-NUM and SPIS-UI), without having the necessity to load the whole components of the other parts.

Instruments are added to the simulation after the UI2NUM phase. Each type of instruments needs input parameters called *InstrumentParameters*. After calling the default constructor, the default mandatory parameters are built at NUM level with default values by a call to *defaultMandatoryParameters()* which returns the list of parameters that must be initialized. They are monitored to the user which can update them at UI level through the instrument wizard. They are then checked NUM level in order to verify the compatibility of the detectors with the simulation (as e.g. not possible to monitor a population which does

not exist). Messages are monitored in the console or in the terminal shell indicating which updates shall be done BEFORE the simulation is run (otherwise, an Exception is thrown and the simulation crashed).
The number and type of mandatory parameters depends on the instrument itself. Description of each parameter is given in the wizard using the information provided in NUM.

Briefly, the collaboration of SPIS packages is the following:
SPIS-UI:

- builds the simulation from the GUI framework data (as usual)
- converts the simulation from UI to NUM
- instantiates instruments using the minimal constructor (passed variables: Id and name of the instrument, the simulation itself)

SPIS-NUM:

- informs UI of the possibilities offered in term of instrument input parameters (with default values)

SPIS-UI:

- invites the user to set the mandatory parameters
- calls the initialize method coded at NUM level using the data passed by the user at GUI level

SPIS-NUM:

- initializes the instrument
- informs UI of possible invalid instrument parameter, in which case an InvalidInstrumentException is thrown (as eg no possibility to measure secondary electron if this population was not created before).

SPIS-UI:

- asks the user to change parameters if necessary
- adds the instruments to the numerical simulation core
- starts the numerical integration

SPIS-NUM:

- runs the simulation including,
- performs the pre-defined or user requested prompt measurements

SPIS-UI:

- monitors the results in different panels
- gets user notifications of instrument modification (performMeasure)
- updates the characteristics of the instrument

3.2.2.4. Implementation classes

In this project, three classes implement the interface ***Instrument***: the ***ParticleDetector***, the ***VirtualParticleDetector*** and the ***PlasmaSensor***.

- The *ParticleDetector* provides mainly the particle distribution functions on dedicated **spacecraft surfaces**. They basically rely on a Test Particle (TP) method which consists in calculating the particle trajectories in a frozen electromagnetic field, by a series of forward and backward tracking. The ***LangmuirProbe*** instrument extends the *ParticleDetecto*r class by introducing IV sweep coupled with TP.
- The *VirtualParticleDetector* has basically the same architecture except it performs the TP onto surfaces outside the spacecraft (**virtual surfaces**). They do not interact with the plasma and spacecraft dynamics.

- The *PlasmaSensor* class provides regularly the time evolution of scalars (or more complex data such as distribution functions or linear data) in the plasma volume domain during the simulation (such as potential, density). It is mainly used to check the convergence of the plasma dynamics at a **specific location of the plasma volume** (not on the spacecraft). They do not interact with the plasma and spacecraft dynamics.

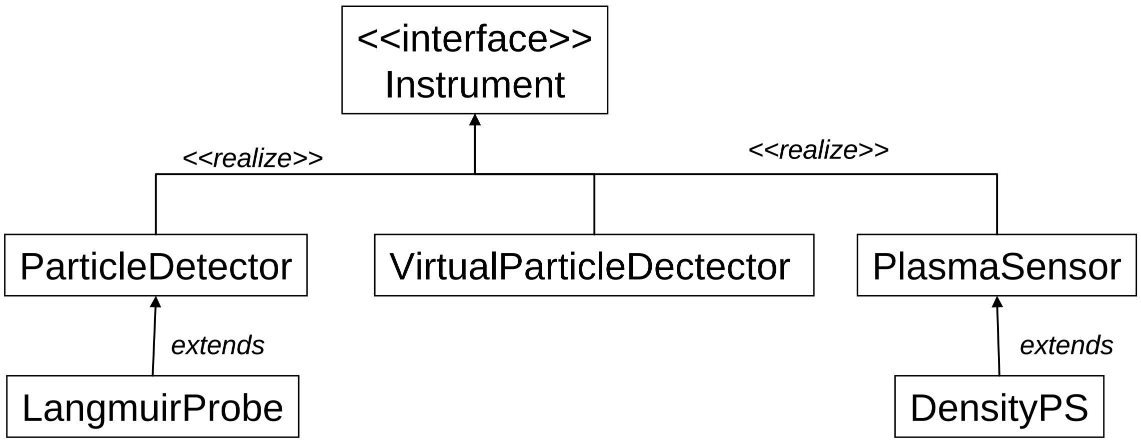


Figure 13 - Inheritance tree of the Instrument interface. Detailed implementation involves intermediate classes.

3.2.3. Particle detectors

The main objective of the particle detector is to provide detailed information about one population simulated in SPIS. The idea is to mimic the measurements done by real particle detector on board of a spacecraft. Usually, the difficulties of this kind of measurement in numerical simulation comes from the multi-scale problematic, i.e. a simulation box for a spacecraft in on the order of several tens of meters but the detection area of a particle detector is on the order of magnitude of millimeters. Furthermore, on this detector, we want to obtain enrich data in comparison to other surfaces. On a standard surface in SPIS, the most important data is the flux of particle to compute the potential equilibrium but the statistical accuracy is rapidly attained (with 100 macro-particles, the flux computed has a statistical error of 10%). On detector surface, the objective is to compute a three dimensional distribution function of velocity thus the number of macro-particles should be larger than on a standard surface.

There are two solutions in order to control the statistic on the detector surface. The first solution is to optimize the PIC simulation in order to increase the statistics of the particle collected by the detector surface. But, these methods are known to have a limited effect and a slow convergence toward the desired statistics in this kind of situations. This method is also used to increase the statistics more globally in volume but it is not sufficiently efficient for particle detector. The second solution is to use a backtracking technique to calculate the distribution function. The drawbacks are that this technique can be used only for non-collisionnal species and it assumes that the situation is quasi-stationary for the particle (i.e. the potential map does not change a lot during the particle path). This technique has been selected for the particle detectors. It has the advantage that the distribution function statistics on the detector can be fixed independently from the PIC simulation.

The general overview of the Particle Detector interaction with the simulation, from user point of view, is depicted in Figure 14. When the current time step of the simulation reaches the instant of an instrument measurement, the particle detection is performed. The result of this observation (referred as “Obs”) is provided to the user, with possibility to reset the detector parameters and re-launch the calculation (to obtain better statistics!). Once the user is satisfied, he notifies the Instrument through the GUI that the simulation can be continued.

Remark: In the case the user does not want to interact with the PD or wait to long, the observation is considered as successful and the simulation is continued. This permits to resume the simulation without spending more time.

Another development consists in performing an observation on user demand. The signal from the user is collected in the GUI and passed to the Instrument, which proceed to the launch of the particle detection once the current plasma-spacecraft integration loop step is completed.

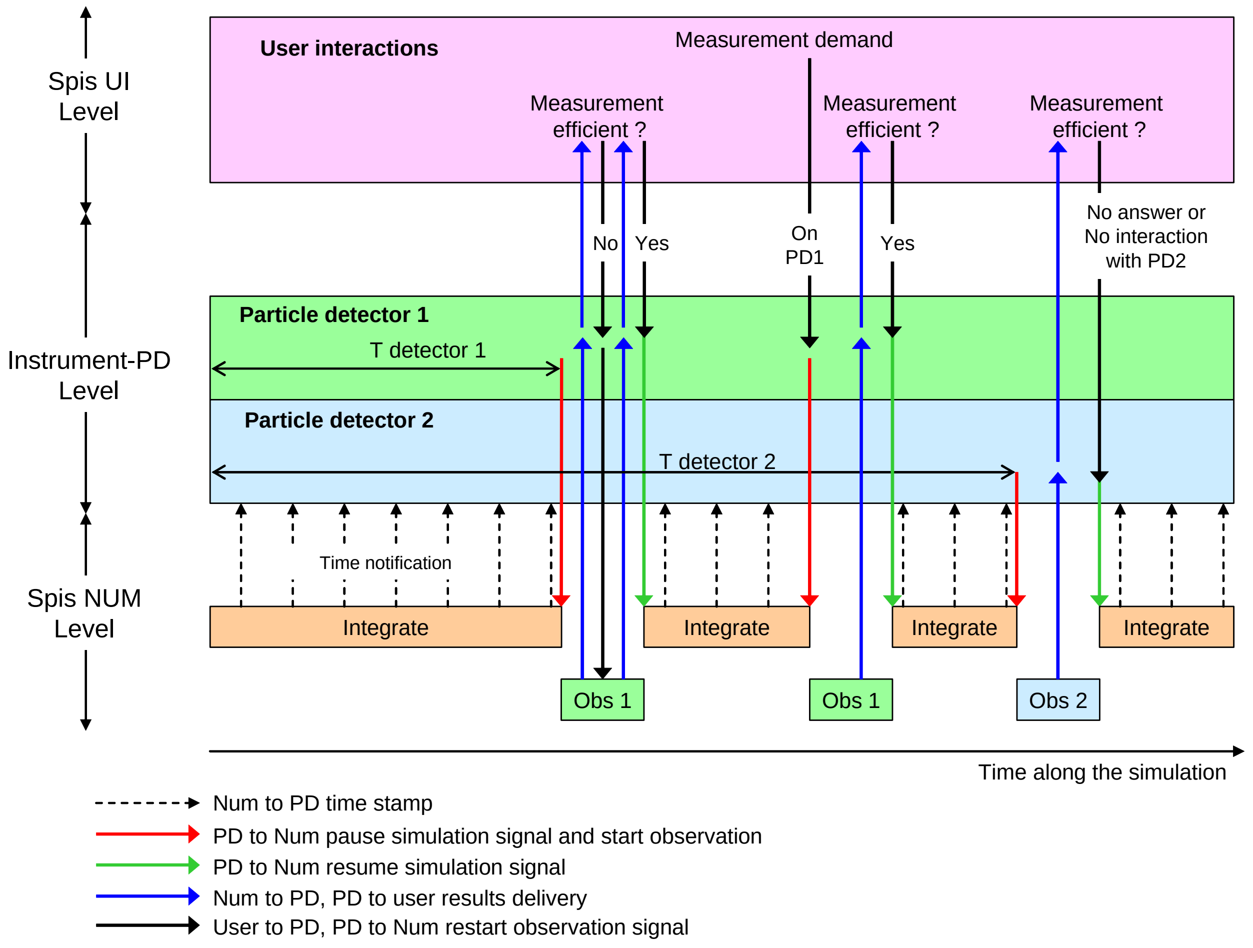
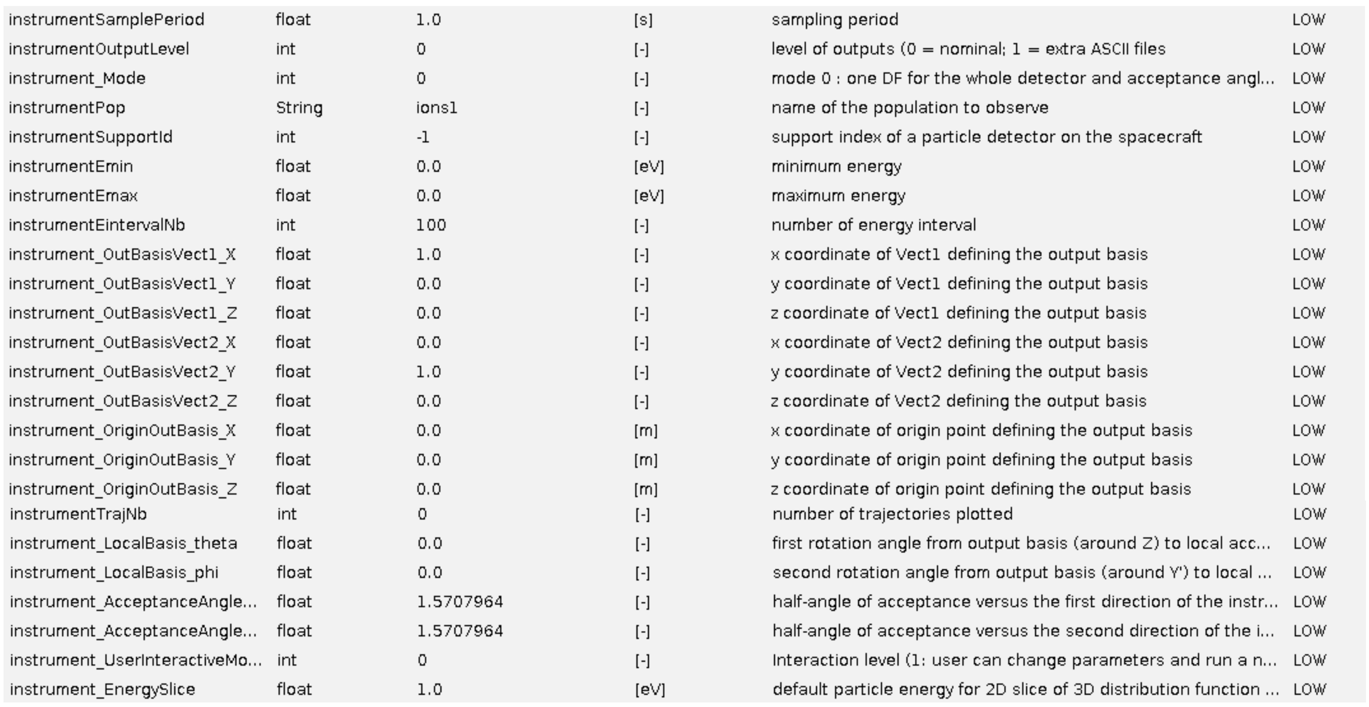


Figure 14 - Control of the simulation and particle detector observations at user/UI, Instrument/PD and NUM levels

3.2.3.1. *Input parameters*



instrumentSamplePeriod	float	1.0	[s]	sampling period	LOW
instrumentOutputLevel	int	0	[-]	level of outputs (0 = nominal; 1 = extra ASCII files	LOW
instrument_Mode	int	0	[-]	mode 0 : one DF for the whole detector and acceptance angl...	LOW
instrumentPop	String	ions1	[-]	name of the population to observe	LOW
instrumentSupportId	int	-1	[-]	support index of a particle detector on the spacecraft	LOW
instrumentEmin	float	0.0	[eV]	minimum energy	LOW
instrumentEmax	float	0.0	[eV]	maximum energy	LOW
instrumentEintervalNb	int	100	[-]	number of energy interval	LOW
instrument_OutBasisVect1_X	float	1.0	[-]	x coordinate of Vect1 defining the output basis	LOW
instrument_OutBasisVect1_Y	float	0.0	[-]	y coordinate of Vect1 defining the output basis	LOW
instrument_OutBasisVect1_Z	float	0.0	[-]	z coordinate of Vect1 defining the output basis	LOW
instrument_OutBasisVect2_X	float	0.0	[-]	x coordinate of Vect2 defining the output basis	LOW
instrument_OutBasisVect2_Y	float	1.0	[-]	y coordinate of Vect2 defining the output basis	LOW
instrument_OutBasisVect2_Z	float	0.0	[-]	z coordinate of Vect2 defining the output basis	LOW
instrument_OriginOutBasis_X	float	0.0	[m]	x coordinate of origin point defining the output basis	LOW
instrument_OriginOutBasis_Y	float	0.0	[m]	y coordinate of origin point defining the output basis	LOW
instrument_OriginOutBasis_Z	float	0.0	[m]	z coordinate of origin point defining the output basis	LOW
instrumentTrajNb	int	0	[-]	number of trajectories plotted	LOW
instrument_LocalBasis_theta	float	0.0	[-]	first rotation angle from output basis (around Z) to local acc...	LOW
instrument_LocalBasis_phi	float	0.0	[-]	second rotation angle from output basis (around Y') to local ...	LOW
instrument_AcceptanceAngle...	float	1.5707964	[-]	half-angle of acceptance versus the first direction of the instr...	LOW
instrument_AcceptanceAngle...	float	1.5707964	[-]	half-angle of acceptance versus the second direction of the i...	LOW
instrument_UserInteractiveMo...	int	0	[-]	Interaction level (1: user can change parameters and run a n...	LOW
instrument_EnergySlice	float	1.0	[eV]	default particle energy for 2D slice of 3D distribution function ...	LOW

Figure 15 - Mandatory parameter for Particle Detectors

In the interface dedicated to instruments, all the mandatory parameters represented in the previous figure are set with default values to be modified by the user. These parameters are updated after each change using the button.

The "*instrumentSamplingPeriod*" parameter defines the period of particle detector measurements in seconds. The SPIS simulation will automatically adapt the time step of the simulation to perform the measurement at the exact requested period. The user may also use the interactive capabilities of instruments and demand a measurement by clicking on . After this live measurement, next measurements will be done with the same period as initially defined.

The "*instrumentPop*" parameter defines the name of the population to backtrack. This population must exist in the simulation but there is no constraint in its type. The backtracking indifferently works for *PICVolDistrib* or *GlobalMaxwellBoltzmannVolDistrid* and for all the volume distributions available in SPIS 5. If the population does not exist, an error message will appear as a popup or in the console at the instrument creation. At the message occurrence, the list of existing population names will be displayed and the user will be invited to change the value of this parameter.

The "*instrumentSupportId*" parameter defines the id of the surface where the instrument is plugged. It could be noticed that several instruments can hold on the same support.

Two modes are implemented for the particles detector. They are controlled by the parameter "*instrumentMode*".

- **Mode 0** is the single distribution function mode. A single but accurate distribution function is calculated for the whole detector surface. The detector acceptance angle and orientation are taken into account (i.e. "*instrument_OutputBasisVect*" parameters, "*instrument_LocalBasis_theta"* and "*instrument_LocalBasis_Phi"* for the orientation of the detector and "*instrument_AnccceptanceAngle"* parameters for the acceptance angles). This mode is well adapted to planar (or quasi-planar) detectors and only approximate in case of radius of curvature.
- **Mode 1** is the multiple distribution functions mode. One distribution function is calculated for each surface element of the detector. To preserve the computational costs, the statistics used in TP method is lower than in mode 0 and the distribution function is a bit noisier. The interest of the mode 1 is however to obtain an accurate value of the flux on a non-planar surface. The acceptance angle and the orientation are not defined in this case (the full velocity space is tracked). This mode is interesting for Langmuir Probe, especially.

The "*instrumentEmin*" and "*instrumentEmax*" parameters define the instrument energy range. NB: this is used both in mode 0 and in mode 1; thus applied to calculate both the flux and the currents too.

The "*instrumentOutputLevel*" parameter defines the verbosity of the detector outputs (see output section).
The "*instrumentEintervalNbr*" parameter corresponds to the number of energy intervals used to plot the results.
The "*instrumentOriginOutputs*" parameters are used to define the origin Cartesian coordinates of the results generated in ASCII file (list of particles with their position).
The "*instrument_EnergySlice*" parameter is the energy used to perform an angular 2D slice of the distribution function.
The "*instrumentTrajNbr*" parameter is the maximal number of trajectories plotted from the detector at each measurement.

3.2.3.2. Detector orientation

WARNING: This section concerns only the particles detectors created in mode 0.

First, the reference basis (X0,Y0,Z0) used to plot the instrument results (as e.g. angular velocity distribution functions, see paragraph 3.2.3.4) is determined by the user, defining two vectors "Vect1" and "Vect2" with the instrument mandatory parameters "*instrument_OutputBasisVect*". The vector coordinates are defined in the GMSH basis used by SPIS for the CAD models:

- Vect1 is used as X0,
- Vect2 as Y0
- Z0 is automatically deduced from X0 and Y0 to form an orthogonal direct basis

A basis common to several detectors may be the simplest way to compare the distribution function from different populations, different locations or different detection surfaces of a same real detector.

The detector orientation is defined by two rotations in this reference basis. The detector local basis (Xd,Yd,Zd) is determined by the user through 2 rotation angles (see next figure):

- Rotation of θ_d around Z0 to obtain the intermediary basis (X', Y', Z'=Zo)
- Rotation of φ_d around Y' to obtain the final local basis (X_d, Y_d=Y', Z_d)

By convention, Z_d is a vector normal to the detector surface pointing inside the detector surface.

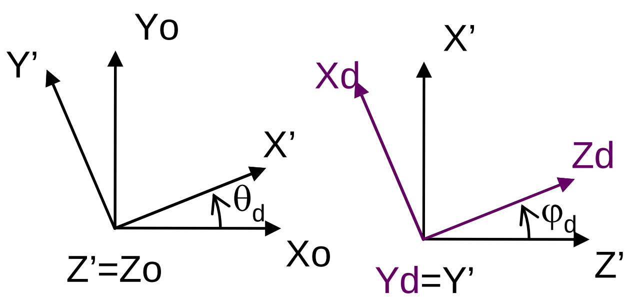


Figure 16 – Definition of the detector orientation

The detector acceptance angles ±α (*instrument_AcceptanceAngle_Alpha*) and ±β (*instrument_AcceptanceAngle_Beta*) are defined by the user in the detector basis (Xd,Yd,Zd).

- ±α around Zd in the plane (Xd, Zd)
- ±β around Zd in the plane (Yd, Zd)

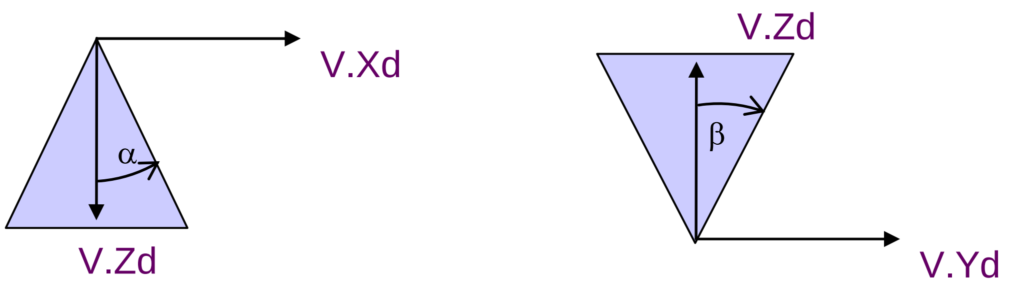


Figure 17 – Definition of acceptance angles

In the next figure, an example of SWA detector configuration is shown. As we can see, the vector Zd is well oriented in the inner direction of the detector.

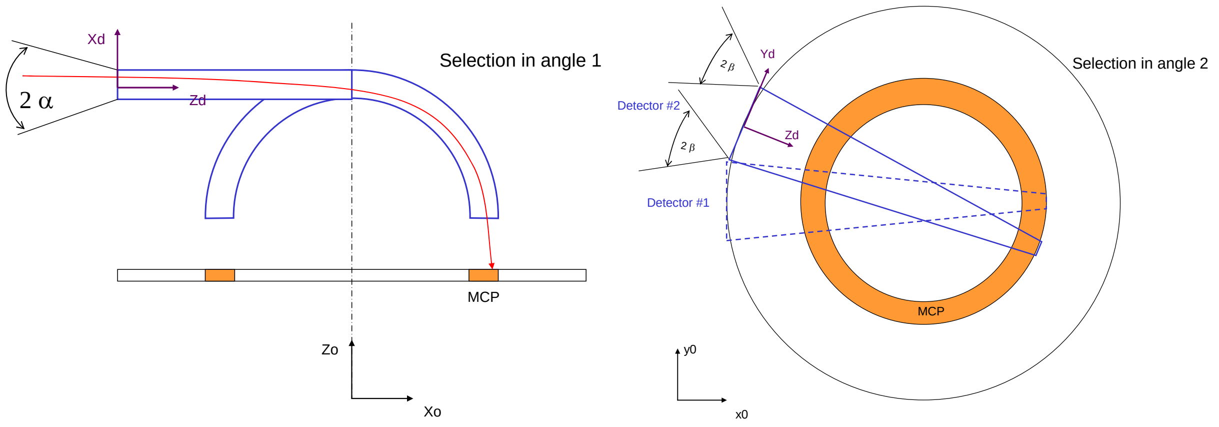


Figure 18 - Example of basis and acceptance angle definition for a SWA detector elementary surface

3.2.3.3. Measurement method

The **backtracking technique** is based on the Liouville's theorem that states the distribution function value is constant along a particle trajectory.

An exact calculation of the distribution function on the detector implies to compute an infinity of trajectories to cover all the velocity space. Here, we discretize the velocity space on the detector into Cartesian velocity volume elements of volume dvx.dvy.dvz. A velocity volume element is a slab between (vx,vy,vz) and (vx+dx,vy+dvy,vz+dz). In each slab, the value of the distribution function is unique. In some other backtracking codes, the value of the distribution function is computed at the center of the element. The test particle has an initial velocity of (vx+dvx/2, vy+dvy/2, vz+dvz/2). Or it could be computed at the corner of the volume element. These methods are quite efficient for very regular distribution functions but fail in the general cases. For example, this determinist method fails when there is a strong selection in energy by the detector. In SPIS, **a Monte Carlo technique** is preferred.

In each slab, a certain number of test particles is created (defined by the user – in the order of 10). For each test particles an initial velocity is randomly sorted between (vx,vy,vz) and (vx+dx,vy+dvy,vz+dz). The value of the distribution function in the elementary volume results from an average of the distribution function computed for all the test trajectories (existing and not existing). For each trajectory:

- If the trajectory does not exist, the value of the distribution function is zero. An example of non-existing trajectory is when an environment particle is backtracked from the detector to a spacecraft surface. A second example of non-existing trajectory is when a secondary electron is backtracked from the detector to the external environment boundary.
- If the trajectory exists, the initial value of the distribution function is computed and applied to the velocity distribution function on the detector surface.

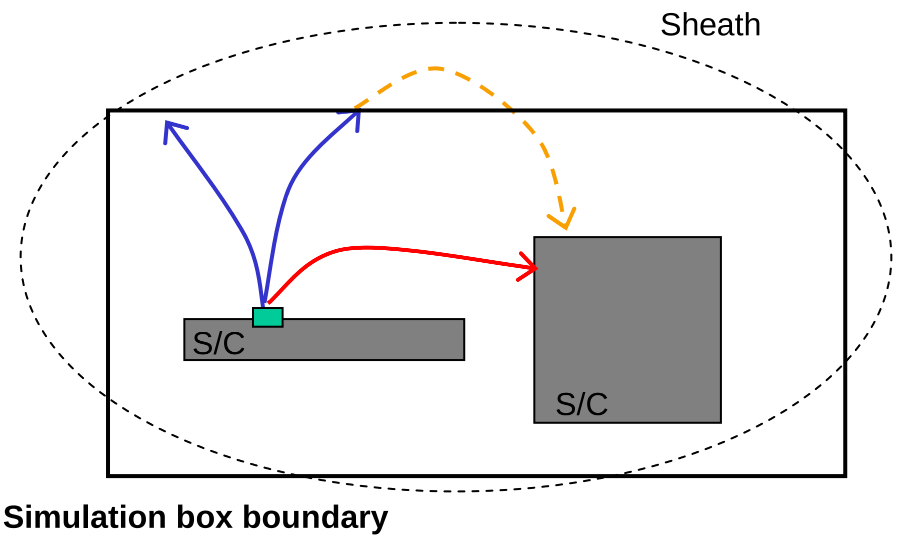


Figure 19 – Backtracking principle from a detector

The backtracking algorithm quality is based essentially on the discretization quality of the velocity space. But most of the time the user can not estimate an accurate range for particle velocity detection, thus the velocity domain to backtrack is very large. Performing an *a priori* discretization would imply to know approximately the distribution function results, which is not generally the case. The chosen solution is to use an **adaptative meshing** of the velocity space volume.

In order to adapt the mesh in course of the backtracking, an OcTree technique is used to store the value of the distribution function. The principle of the OcTree algorithm is to start with a single slab representing the whole velocity space volume and then to successively split in eight sub-volumes (length divided by two in the three directions) the volumes you want to refine. This algorithm is represented in the next figure. It is in fact a dichotomy algorithm for mesh splitting. Thus the convergence is very fast, in 2^N in each direction with N the number of splitting. In 20 iterations, the precision of discretization in one direction can be dvx/10^6!

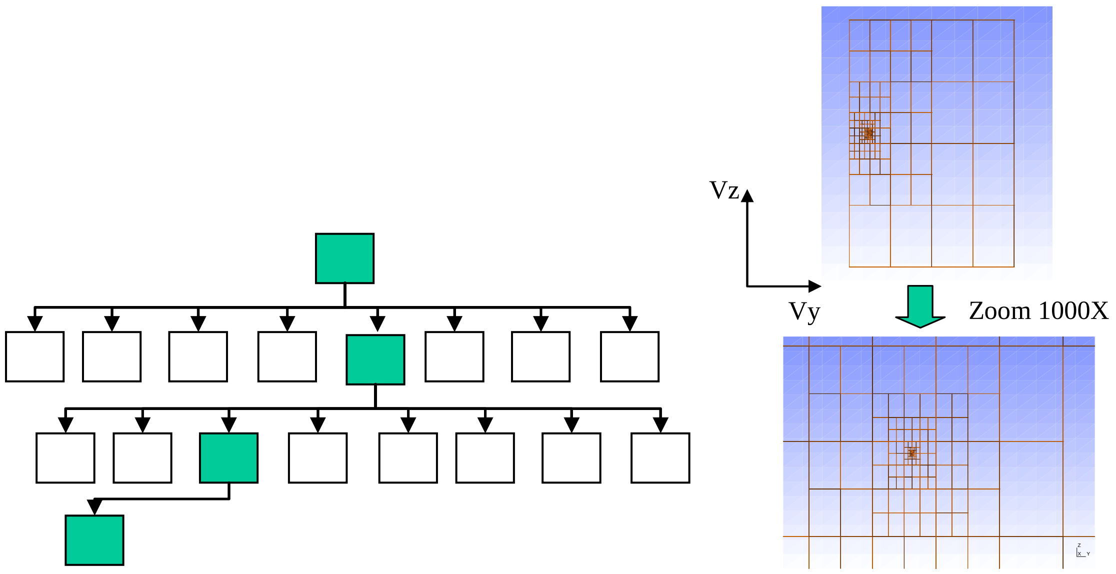


Figure 20 – OcTree architecture and the corresponding distribution function discretization in the velocity space

The next step of the algorithm is the selection of volume elements which are interesting to split. An heuristic of optimization is used. This heuristic is based on the fact that two kinds of volume elements are very interesting to improve:

- Volume elements in which the distribution function value are the highest. In these elements, it is always interesting to increase the precision because it automatically leads to a better estimation of the distribution function moments.
- Elements in which the test particle method computes very different distribution function values. As the value is an average, it is interesting to split this volume to exactly compute for example the exact position of a distribution function gradient. Conversely, when all the test particles lead to the same value of the distribution function, it is reasonable to think that there is no need to refine this volume even if the value of the distribution function is the highest.
- Elements which have neighbors with a greater splitting level. It permits to propagate splitting refinement to neighbors and results in more homogeneous meshes through a diffusion mechanism.

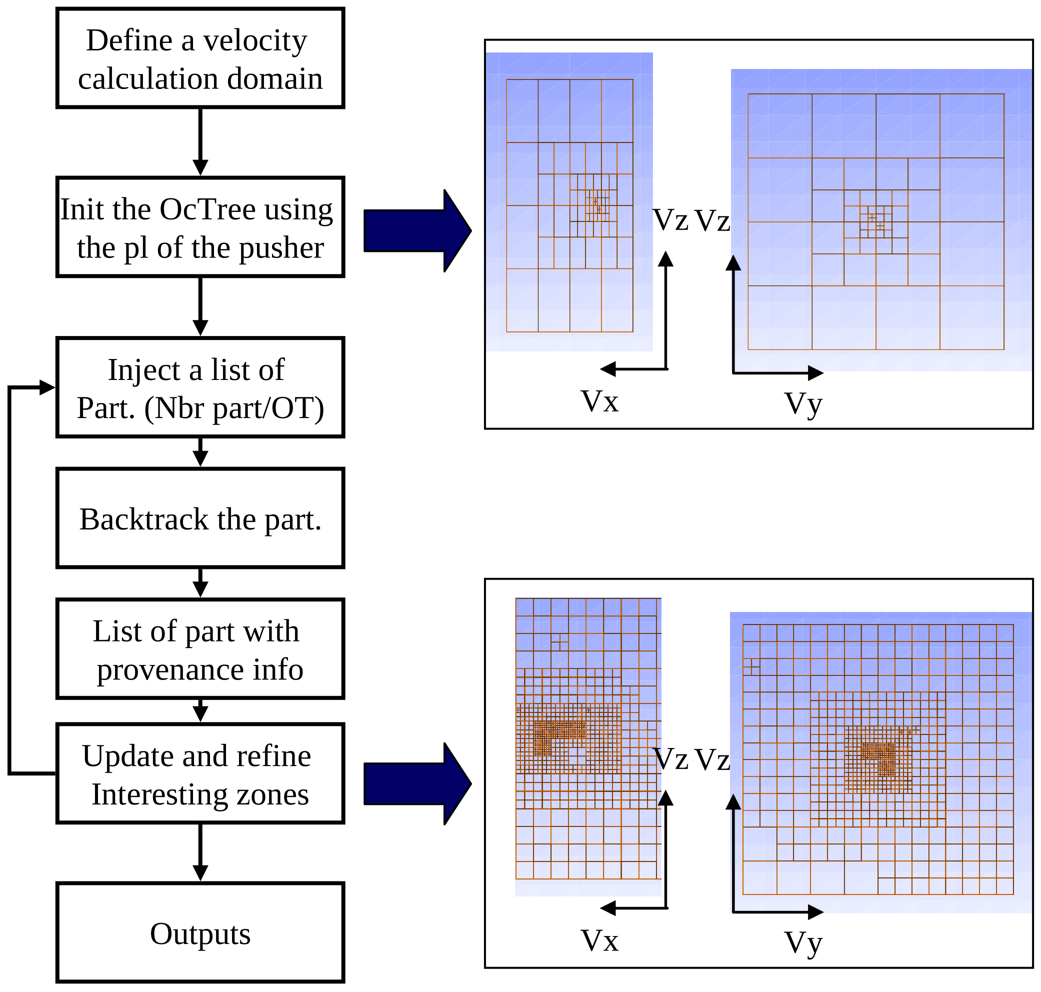

Define a velocity calculation domain
Init the OcTree using the pl of the pusher
Inject a list of Part. (Nbr part/OT)
Backtrack the part.
List of part with provenance info
Update and refine Interesting zones
Outputs
Vz Vz
Vx
Vy
Vz Vz
Vx
Vy

Figure 21 – Optimization heuristic for the distribution function discretization

3.2.3.4. Outputs

After each measurement, a *ParticleMeasurementResult* output is collected and presented to the user at UI level in the instrument wizard. This is done by methods giving access to:

- Distribution function as a function of the energy. It is performed at detection on the instrument and at injection (on the external boundary for ambient populations and on the spacecraft for secondaries)
- Differential flux as a function of the energy (at detection on instrument)
- Slice of the distribution function at a given energy as a function of elevation (phi) and azimuth (theta) angles defined in the *OutBasis* referential frame (defined at detection and injection).
- Slice of the differential flux at a given energy as a function of elevation (phi) and azimuth (theta) angles defined in the *OutBasis* referential frame (defined at detection).
- Spectrogram of the energy distribution function vs. time (at detection and injection)
- Spectrogram of the differential flux as a function of the energy vs. time (at detection)

In addition, 2D Data Fields (current density at detection and injection) are generated.

When the parameter "*instrumentOutputLevel*" is 1, additional ASCII files are created in the NumKernel/Output/ repository:

- "spis.Util.Instrument.ParticleDetector1_Moment_at_XXXXs.txt" presents the different values of the first three moments associated to the distribution function on the detector, the flux distribution function and the distribution function at the particle source (environment or spacecraft).
- "spis.Util.Instrument.ParticleDetector1_Differential_Flux_and_Energy_at_XXXXs.txt" presents the distribution of flux, the distribution of energy and the initial distribution of energy as a function of energy.
- "spis.Util.Instrument.ParticleDetector1_Total_Current_Collection.txt" is the value of the total current on the detector as a function of time (one value by measurement).
- "spis.Util.Instrument.ParticleDetector1_2D_DifferentialFlux_at_t=XXXXs.txt" presents the 3D tabulated values of the flux distribution function on the detector in the output frame (Cartesian coordinate vx,vy,vz).
- "spis.Util.Instrument.ParticleDetector1_2D_Angular_DifferentialFlux_at_t=XXXXs.txt" presents the 3D tabulated values of the flux distribution function on the detector in the output frame (spherical coordinate E, theta, phi).
- "spis.Util.Instrument.ParticleDetector1_Velocity2DF_at_t=XXXXs.txt" presents the 3D tabulated values of the velocity distribution function on the detector in the output frame (Cartesian coordinate vx,vy,vz).
- "spis.Util.Instrument.ParticleDetector1_Angular2DF_at_t=XXXXs.txt" presents the 3D tabulated values of the velocity distribution function on the detector in the output frame (spherical coordinate E, theta, phi).
- "spis.Util.Instrument.ParticleDetector1_InitialVelocity2DF_at_t=XXXXs.txt" presents the 3D tabulated values of the velocity distribution function on the surface source of particles in the output frame (Cartesian coordinate vx,vy,vz).
- "spis.Util.Instrument.ParticleDetector1_Initial_Angular2DF_at_t=XXXXs.txt" presents the 3D tabulated values of the velocity distribution function on the surface source of particles in the output frame (spherical coordinate E, theta, phi).
- "spis.Util.Instrument.ParticleDetector1_3V_Distribution_Function_at_t=XXXXs.msh" is the 3V distribution function on the detector in the OcTree form in the GMS frame (readable in GMSH).
- "spis.Util.Instrument.ParticleDetector1_3V_Differential_Flux_at_t=XXXXs.msh" is the 3V differential flux on the detector in the OcTree form in the GMS frame (readable in GMSH).
- "spis.Util.Instrument.ParticleDetector1_3V_Initial_Distribution_Function_at_t=XXXXs.msh" is the 3V distribution function on the surface source of particles in the OcTree form in the GMS frame (readable in GMSH).
- "spis.Util.Instrument.ParticleDetector1_Particle_List_at_t=XXXXs.msh" is the list of detected particle on the detector in the output frame. There is one line per particles with successively the position on the detector (xD,yD,zD), the velocity on the detector (vxD,vyD,vzD), the flux weight of the particle on the detector (wFD), the position on the particle source (xE,yE,zE), the velocity on the particle source (vxE,vyE,vzE), the flux weight of the particle on the particle source (wFE) and the statistical weight of the particle in volume (w) which is conserved in Liouville theorem.
- map of current on the detectors in mode 1 and where the current is coming from.

Angular distributions f(θ,φ) calculated by SPIS are then given in the reference basis (Xo,Yo,Zo) by defining the velocity vectors in polar coordinates (V_r, θ, φ) as defined in Figure 22:

- Rotation of θ around Zo axis
- Rotation of φ around modified Yo axis according to next figure.

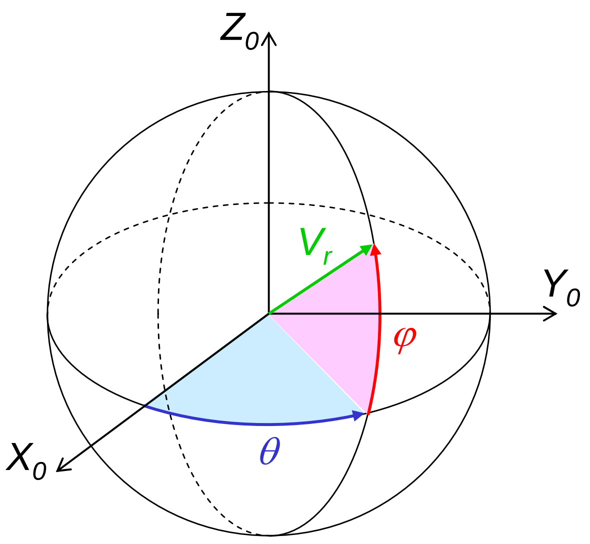


Figure 22 - Definition of velocity polar coordinates (V_r, θ, φ) in the reference basis (Xo,Yo,Zo)

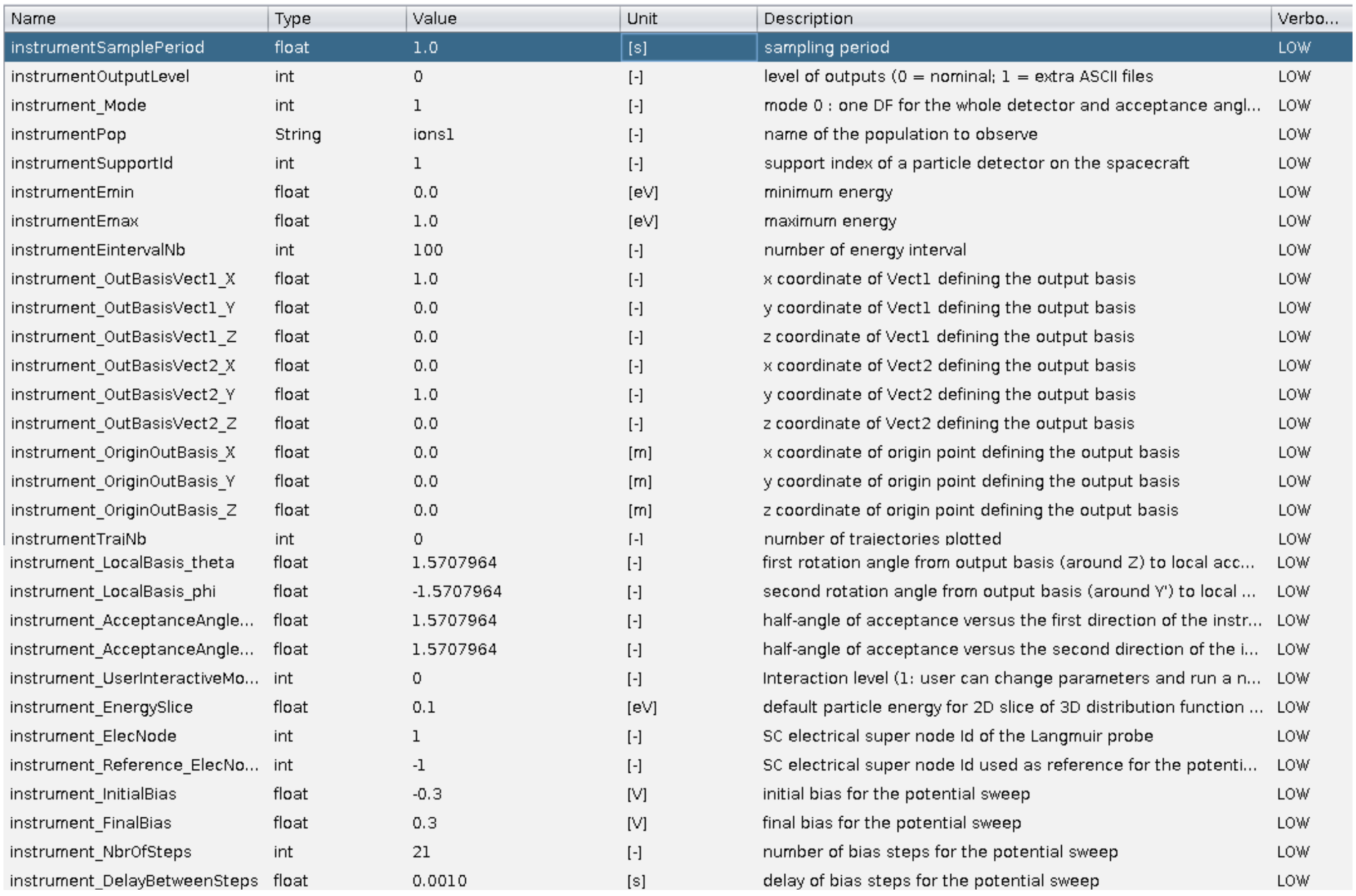
3.2.3.5. User interactive mode

The objective was to propose the user to validate the results and resume the simulation after a measurement or to change the instrument parameters for a new TP run. An instrument listener has been added to the instrument. It notifies UI whenever a measurement has been performed. The user can choose to resume the simulation or re-start the TP **AFTER** updating the parameters. In case of a new TP run, previous results obtained on current measurement are replaced.

3.2.4. Langmuir Probes

The Langmuir Probe design is based on the same principle as Particle Detectors. More precisely, a Langmuir Probe is a particle detector which measures a particle current (mode 1) combined with a voltage sweep applied detector surfaces (through a dedicated transition class).

3.2.4.1. Input parameters



Name	Type	Value	Unit	Description	Verbo...
instrumentSamplePeriod	float	1.0	[s]	sampling period	LOW
instrumentOutputLevel	int	0	[-]	level of outputs (0 = nominal; 1 = extra ASCII files	LOW
instrument_Mode	int	1	[-]	mode 0 : one DF for the whole detector and acceptance angl...	LOW
instrumentPop	String	ions1	[-]	name of the population to observe	LOW
instrumentSupportId	int	1	[-]	support index of a particle detector on the spacecraft	LOW
instrumentEmin	float	0.0	[eV]	minimum energy	LOW
instrumentEmax	float	1.0	[eV]	maximum energy	LOW
instrumentEintervalNb	int	100	[-]	number of energy interval	LOW
instrument_OutBasisVect1_X	float	1.0	[-]	x coordinate of Vect1 defining the output basis	LOW
instrument_OutBasisVect1_Y	float	0.0	[-]	y coordinate of Vect1 defining the output basis	LOW
instrument_OutBasisVect1_Z	float	0.0	[-]	z coordinate of Vect1 defining the output basis	LOW
instrument_OutBasisVect2_X	float	0.0	[-]	x coordinate of Vect2 defining the output basis	LOW
instrument_OutBasisVect2_Y	float	1.0	[-]	y coordinate of Vect2 defining the output basis	LOW
instrument_OutBasisVect2_Z	float	0.0	[-]	z coordinate of Vect2 defining the output basis	LOW
instrument_OriginOutBasis_X	float	0.0	[m]	x coordinate of origin point defining the output basis	LOW
instrument_OriginOutBasis_Y	float	0.0	[m]	y coordinate of origin point defining the output basis	LOW
instrument_OriginOutBasis_Z	float	0.0	[m]	z coordinate of origin point defining the output basis	LOW
instrumentTrajNb	int	0	[-]	number of trajectories plotted	LOW
instrument_LocalBasis_theta	float	1.5707964	[-]	first rotation angle from output basis (around Z) to local acc...	LOW
instrument_LocalBasis_phi	float	-1.5707964	[-]	second rotation angle from output basis (around Y') to local ...	LOW
instrument_AcceptanceAngle...	float	1.5707964	[-]	half-angle of acceptance versus the first direction of the instr...	LOW
instrument_AcceptanceAngle...	float	1.5707964	[-]	half-angle of acceptance versus the second direction of the i...	LOW
instrument_UserInteractiveMo...	int	0	[-]	Interaction level (1: user can change parameters and run a n...	LOW
instrument_EnergySlice	float	0.1	[eV]	default particle energy for 2D slice of 3D distribution function ...	LOW
instrument_ElecNode	int	1	[-]	SC electrical super node Id of the Langmuir probe	LOW
instrument_Reference_ElecNo...	int	-1	[-]	SC electrical super node Id used as reference for the potenti...	LOW
instrument_InitialBias	float	-0.3	[V]	initial bias for the potential sweep	LOW
instrument_FinalBias	float	0.3	[V]	final bias for the potential sweep	LOW
instrument_NbrOfSteps	int	21	[-]	number of bias steps for the potential sweep	LOW
instrument_DelayBetweenSteps	float	0.0010	[s]	delay of bias steps for the potential sweep	LOW

Figure 23 - Mandatory parameter for Langmuir Probes

As the LP is based on a Particle Detector, all the mandatory parameters of a Particle Detector are needed. It can be noted that the calculation mode is not force to 1 but it is recommended to use the LP in this mode ("*instrument_Mode*" parameter).

Additional parameters are also needed in order to configure the voltage sweep Transition.

The "*instrumentSamplingPeriod*" parameter still defines the period of occurrence of a sequence of measurement. In the case of a LP, this period is the duration between two sequences of measurements and V-sweeps. It is hence possible to perform several V-sweep along the simulation, each V-sweep being associated to electrical current measurements.

The "*instrument_DelayBetweenSteps*" parameter defines the delay between two V sweep steps inside each V-sweep sequence. It is the responsibility of the user to insure that there is a sufficient time to perform a complete IV sweep during one sampling period:

"*instrumentSamplingPeriod*" > "*instrument_DelayBetweenSteps*" x ("*instrument_NbrOfSteps*" + 1)

The detector potentials range is defined with the parameters "*instrument_InitialBias*" and "*instrument_FinalBias*". The number of potential steps is defined by "*instrument_NbrOfSteps*".

When "*instrument_ReferenceElecNode*" is set to -1, the LP is in "Ground Mode". The potential of the electrical node corresponding to the parameter "*instrument_ElecNode*" is changed following the IV sweep. The bias is applied in comparison to the ground at infinite (0 V).

When "*instrument_ReferenceElecNode*" is set to a positive value corresponding to an existing electrical node of reference, the LP is in "Floating Mode". The potential of the electrical node defined in the parameter "*instrument_ElecNode*" is changed following the IV sweep, with respect to the potential the reference electrical node.

3.2.4.2. Measurements

The transition associated with each LP controls the potential sweep applied on the selected electrical node. The particle measurement is done after a delay of stabilization of the simulation between two steps (i.e. "*instrument_DelayBetweenSteps*").

First, the measurement of the current at a fixed potential is a standard output of the particle detector. The user can use the particle detector in mode 0 or mode 1. All the standard results after a particle detector measurement are provided to the user. In addition, the current-voltage sweep is built by concatenation of current measurement at each step. ASCII files are provided for each LP and for each measurement sequence of a LP as e.g. “Langmuir probe id2_IVcurve_Seq1".

NB: The LP was created to cover two different situations:

Situation 1: Not floating spacecraft

- the circuit integration shall not be activated (no floating potential)
- no circuit file shall be defined
- the simulation time step shall be fixed (simulationDt < 0)
- The initial potential shall be defined through globalParameters or localParameters
- the LP sweep shall be defined between the infinity -1 and the LP node

Situation 2: LP on a floating spacecraft

- the circuit integration shall be activated (the potentials are floating)
- a circuit file shall be defined
- the simulation time shall be automatic (simulationDt >= 0)
- the LP sweep shall be defined between two node => the node n1 and the LP node n2
- **Warning** the circuit file and the LP can potentially have a concurrent access on the same SC circuit components, which is prohibited As a consequence and example, fixing a potential between node 0 and node 1 and between node 1 and node 2, excludes the possibility to define an IV sweep between node 0 and node 2.

3.2.4.3. User interactive mode

A limited interactive mode is permitted for LP. Only parameters defining the test particle method can be changed (no change of the V-sweep characteristics).

3.2.5. Virtual Particle Detectors

The Virtual Particle Detector (VPD) aims at measuring particle fluxes on virtual surfaces immersed in the computational domain without connection to the boundaries (i.e S/C surface nor external bound). In comparison to the *ParticleDetector* class, the *VirtualParticleDetector* class (VPD) needs a new attribute: a separated surface mesh, here after called Virtual Surface Mesh (VSM).

The VSM is defined at SPIS-UI level and meshed independently. It is composed of 2D triangular mesh elements. Each VSM represents a possible support of VPD and each support can hold several VPD. The VSM and virtual supports are defined and passed to the SPIS-NUM simulation instance before the beginning of the simulation and VSM cannot be loaded within the course of the simulation

SPIS-NUM computes the correspondence between the VSM nodes and the tetrahedra composing the simulation volume mesh. The VSM nodes shall be inside the volume mesh domain.

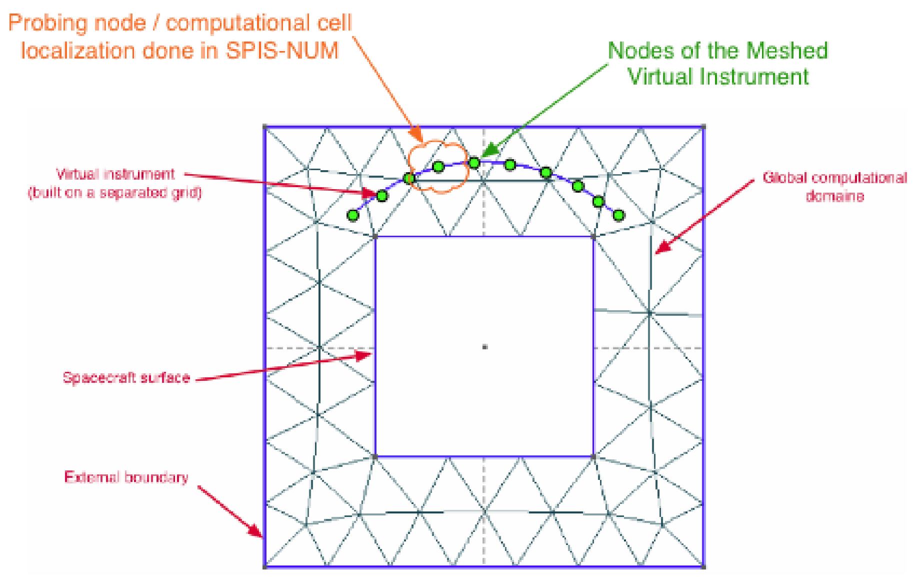
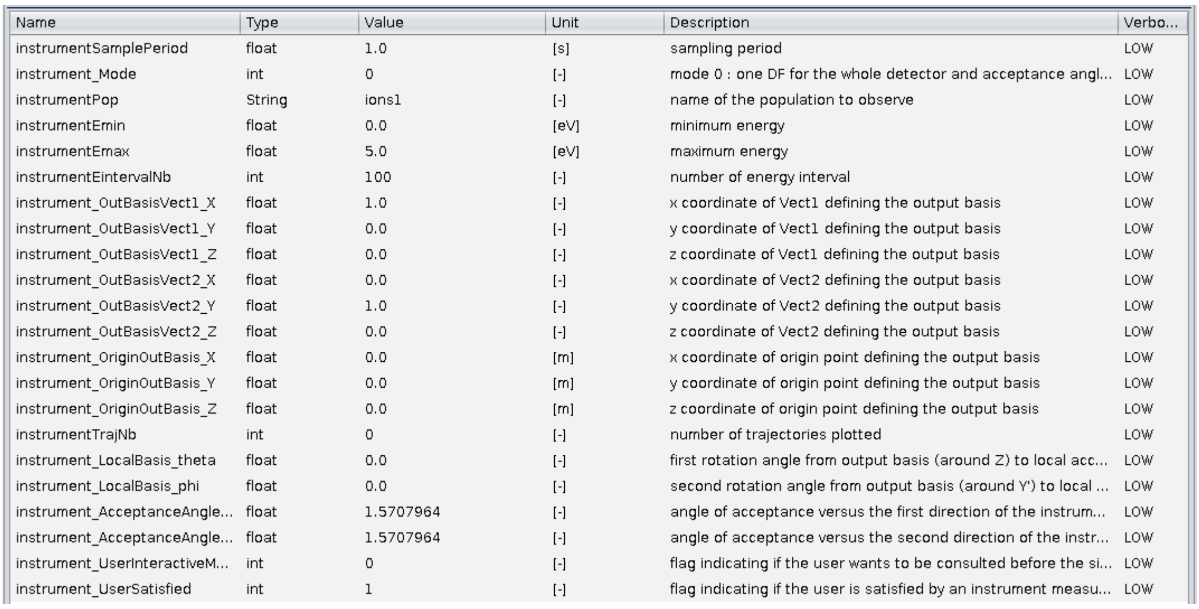

Probing node / computational cell localization done in SPIS-NUM
Nodes of the Meshed Virtual Instrument
Virtual instrument (built on a separated grid)
Global computational domains
Spacecraft surface
External boundary

Figure 24 - Schematical description of a virtual particle detector mesh

3.2.5.1. *Input parameters*



Name	Type	Value	Unit	Description	Verbo...
instrumentSamplePeriod	float	1.0	[s]	sampling period	LOW
instrument_Mode	int	0	[-]	mode 0 : one DF for the whole detector and acceptance angl...	LOW
instrumentPop	String	ions1	[-]	name of the population to observe	LOW
instrumentEmin	float	0.0	[eV]	minimum energy	LOW
instrumentEmax	float	5.0	[eV]	maximum energy	LOW
instrumentEintervalNb	int	100	[-]	number of energy interval	LOW
instrument_OutBasisVect1_X	float	1.0	[-]	x coordinate of Vect1 defining the output basis	LOW
instrument_OutBasisVect1_Y	float	0.0	[-]	y coordinate of Vect1 defining the output basis	LOW
instrument_OutBasisVect1_Z	float	0.0	[-]	z coordinate of Vect1 defining the output basis	LOW
instrument_OutBasisVect2_X	float	0.0	[-]	x coordinate of Vect2 defining the output basis	LOW
instrument_OutBasisVect2_Y	float	1.0	[-]	y coordinate of Vect2 defining the output basis	LOW
instrument_OutBasisVect2_Z	float	0.0	[-]	z coordinate of Vect2 defining the output basis	LOW
instrument_OriginOutBasis_X	float	0.0	[m]	x coordinate of origin point defining the output basis	LOW
instrument_OriginOutBasis_Y	float	0.0	[m]	y coordinate of origin point defining the output basis	LOW
instrument_OriginOutBasis_Z	float	0.0	[m]	z coordinate of origin point defining the output basis	LOW
instrumentTrajNb	int	0	[-]	number of trajectories plotted	LOW
instrument_LocalBasis_theta	float	0.0	[-]	first rotation angle from output basis (around Z) to local acc...	LOW
instrument_LocalBasis_phi	float	0.0	[-]	second rotation angle from output basis (around Y') to local ...	LOW
instrument_AcceptanceAngle...	float	1.5707964	[-]	angle of acceptance versus the first direction of the instrum...	LOW
instrument_AcceptanceAngle...	float	1.5707964	[-]	angle of acceptance versus the second direction of the instr...	LOW
instrument_UserInteractiveM...	int	0	[-]	flag indicating if the user wants to be consulted before the si...	LOW
instrument_UserSatisfied	int	1	[-]	flag indicating if the user is satisfied by an instrument measu...	LOW

Figure 25 - Mandatory parameter for a Virtual Particle Detector

The same parameters used for particle detectors are used for VPD.

3.2.5.2. *Measurements method*

As compared to a particle detector, the only specificity of a VPD is its meshed surface. All the surface distribution functions and all the particle pusher methods have been adapted to VSM.

The trickiest point concerned the injection of test particles in the computational volume. A specific position sampler has been developed to make a connection between the VSM and the calculation volume mesh. To preserve SPIS solvers efficiency, a bufferized position sampler of test particles aims at keeping in memory the link between each virtual surface mesh elements and its neighbors in the volume mesh. This is an important step since injection is performed very often.

3.2.5.3. User interactive mode

Same as a Particle detector.

3.2.6. Plasma Sensors

The purpose of the *PlasmaSensor* class (PS) is to measure all requested physical fields (e.g. density, potential) mapped on the volume computational grid. This measurement is dynamically printed to the screen in the form of time series. This has both scientific and numerical purposes (informing on the simulation convergence). As of today, only basic domains are permitted (point, sphere, lines).

The control of the interaction between the PS and the simulation is described at UI, NUM and user level in Figure 26. During the simulation, NUM delivers regular notifications to the PS instrument, which performs measurement when pre-defined observation times are reached. UI plots them on the screen. On user demand, UI can update the parameters of the PS. Also a measurement can be performed on user demand.

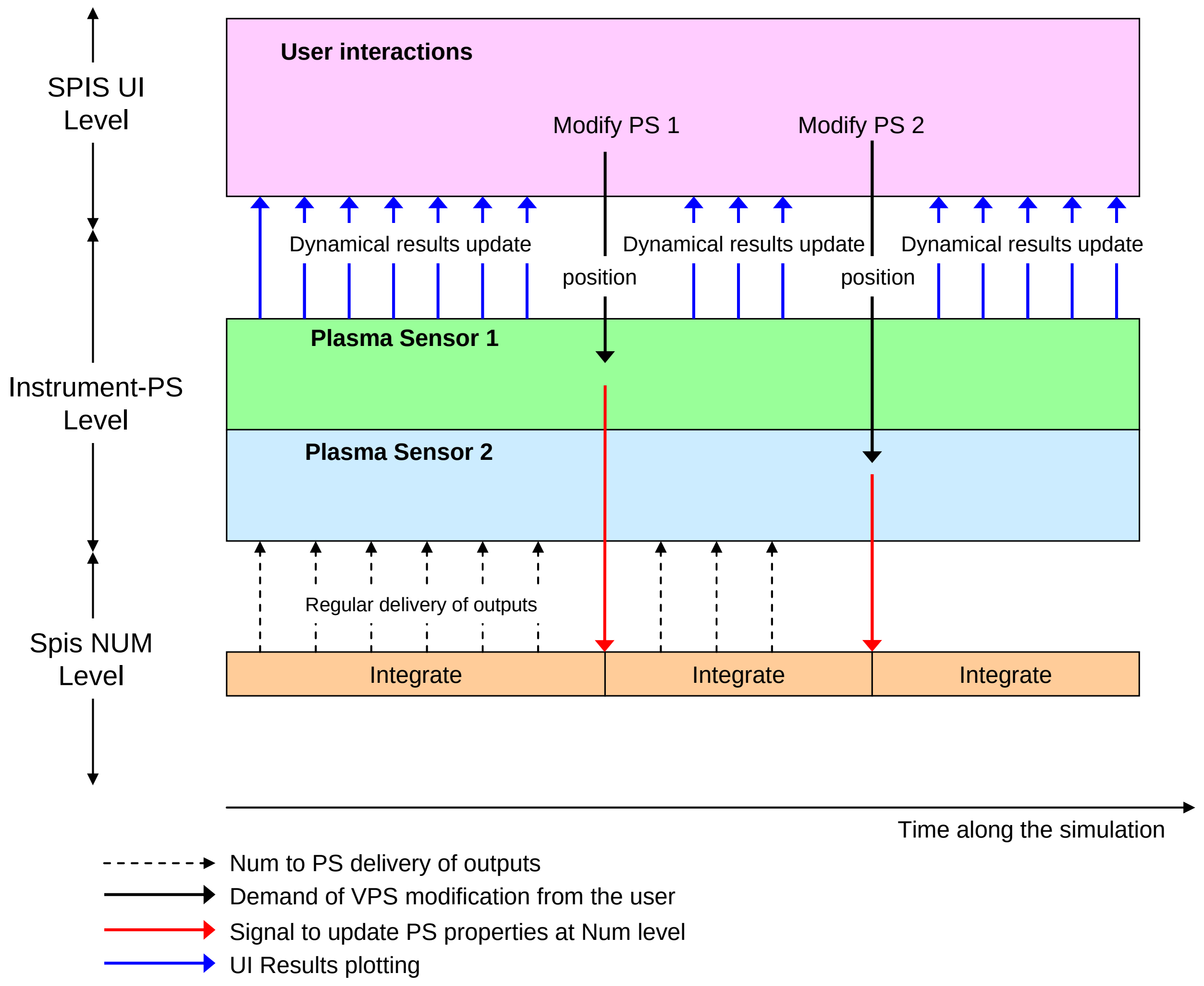
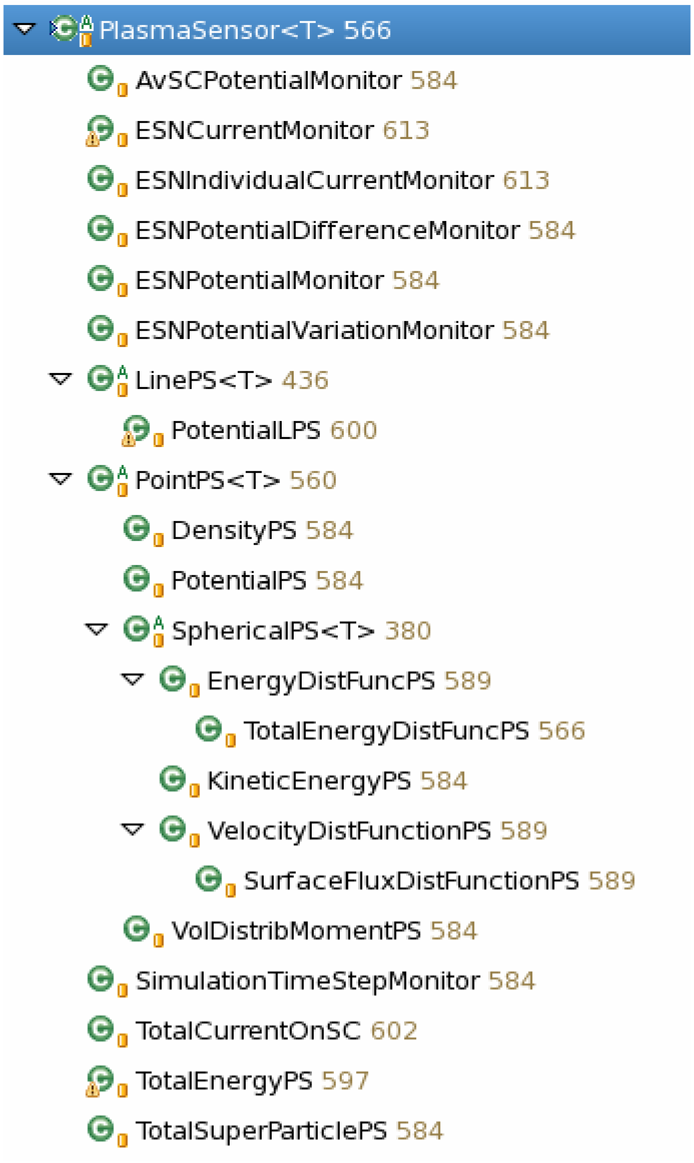


Figure 26 - Control of the simulation and virtual plasma sensor observations at user, UI and NUM levels

The full list of available plasma sensors is summarized in the next tables.

Table 1 - List of plasma sensors sub types



```
PlasmaSensor<T> 566
    AvSCPotentialMonitor 584
    ESNCurrentMonitor 613
    ESNIndividualCurrentMonitor 613
    ESNPotentialDifferenceMonitor 584
    ESNPotentialMonitor 584
    ESNPotentialVariationMonitor 584
    LinePS<T> 436
        PotentialLPS 600
    PointPS<T> 560
        DensityPS 584
        PotentialPS 584
        SphericalPS<T> 380
            EnergyDistFuncPS 589
                TotalEnergyDistFuncPS 566
            KineticEnergyPS 584
            VelocityDistFunctionPS 589
                SurfaceFluxDistFunctionPS 589
        VolDistribMomentPS 584
    SimulationTimeStepMonitor 584
    TotalCurrentOnSC 602
    TotalEnergyPS 597
    TotalSuperParticlePS 584
```

Table 2 - Input parameters for plasma sensors

Instrument Type	Quantity	Result Type	Position	Sampling Period	Sampling Duration	Pop.	Radius	Emin	Emax	Nb of E intervals	OutBasis	ESN Id
PotentialPS	Potential	DimScal	X	X	X							
DensityPS	pop density	DimScal	X	X	X	X						
EnergyDistFuncPS	Kinetic energy histogram	ParticleMeasurementResult	X	X	X	X	X	X	X	X		
TotalEnergyDistFuncPS	Total energy histogram	ParticleMeasurementResult	X	X	X	X	X	X	X	X		
KineticEnergyPS	Mean Kinetic energy	DimScal	X	X	X	X	X					
VelocityDistFunctionPS	3D velocity DF	ParticleMeasurementResult	X	X	X	X	X	X	X	X	X	
SurfaceFluxDistFunctionPS		ParticleMeasurementResult	X	X	X	X	X	X	X	X	X	
TotalEnergyPS	Total pop energy in domain	DimScal		X	X	X						
TotalSuperParticlePS	Total nb pop super particles	DimScal		X	X	X						
AvSCPotentialMonitor	Mean SC potential	DimScal		X	X							

ESNCurrentMonitor	Total current on ESN	DimScal		X	X							X
ESNIndividualCurrentMonitor	Detailed currents	DimScal		X	X							X
ESNPotentialMonitor	Potential of ESN	DimScal		X	X							X
ESNPotentialDifferenceMonitor	Diff pot vs local ground	DimScal		X	X							X
ESNPotentialVariationMonitor	dV/dt of ESN	DimScal		X	X							X

Remarks:

- VelocityDistFunctionPS is the 3V distribution function defined in the referential frame *OutBasis*
- SurfaceFluxDistFunctionPS is the 3V differential flux of a population passing through an oriented (normal in OutBasis first vector) disk of radius *Radius*

3.3. Semi-transparent grids

From [SRD], semi-transparent grids (STG) are thin spacecraft surfaces bounded by two plasma volumes. They collect only a fraction of particles passing through them. They can emit secondary particles (from electron, proton and photon impact).

3.3.1. CAD modeling

At UI level, a STG is characterized by:

- The classical plasma volume mesh hereafter called volMesh (possibly a combination of several volume meshes) with the following extra information under the form of DataFields:
    - Flags on surface mesh elements of volMesh belonging to a STG: surface flag indicating this is a STG:
        - Flag of the STG
        - Id of the STG,
        - transparency coefficient normalized between 0 and 1,
    - For each surface mesh element of volMesh belonging to a STG, the correspondence with the index of the surface elements in the two side surface mesh of the STG (i.e couple of indexes);
- An additional surface mesh
    - This surface mesh is composed of two sides,
    - Side A is the duplicate of face B but with surface meshes oriented in the opposite direction
    - Side A et side B are concatenated in order to form a single surface mesh
- A single surface mesh grid is defined for all the STG

The surface meshes are passed from SPIS-UI to SPIS-NUM, as the legacy volume and surface meshes.

UNCLASSIFIED

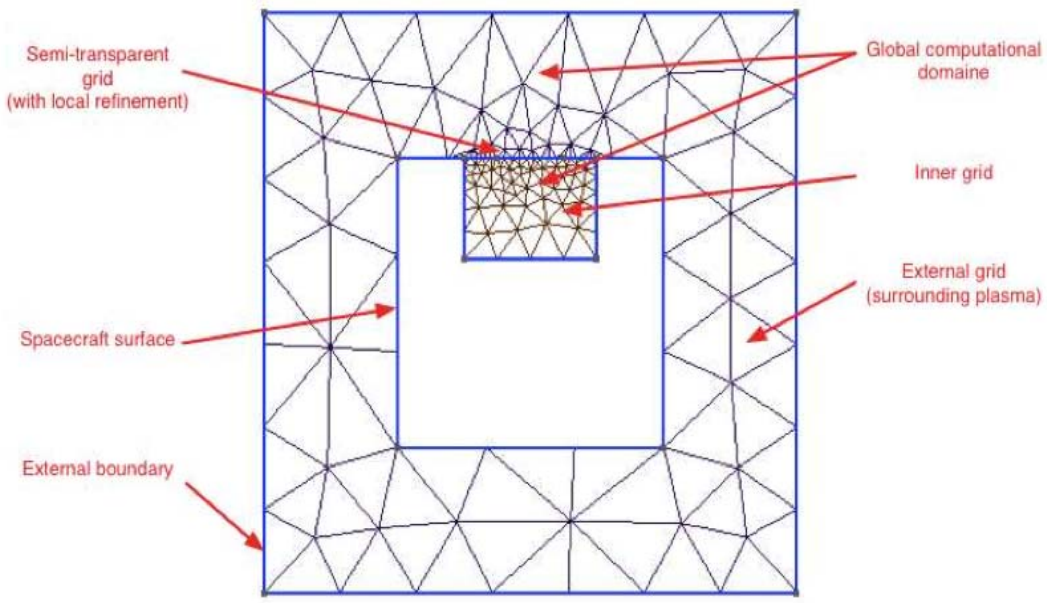

Semi-transparent grid (with local refinement)
Global computational domaine
Inner grid
External grid (surrounding plasma)
Spacecraft surface
External boundary

Figure 27 - Example of Cad modelling of a STG inside the computational domain

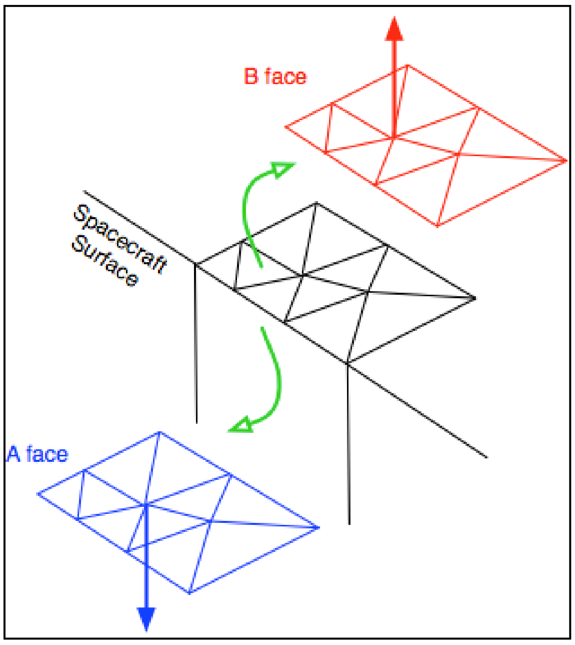
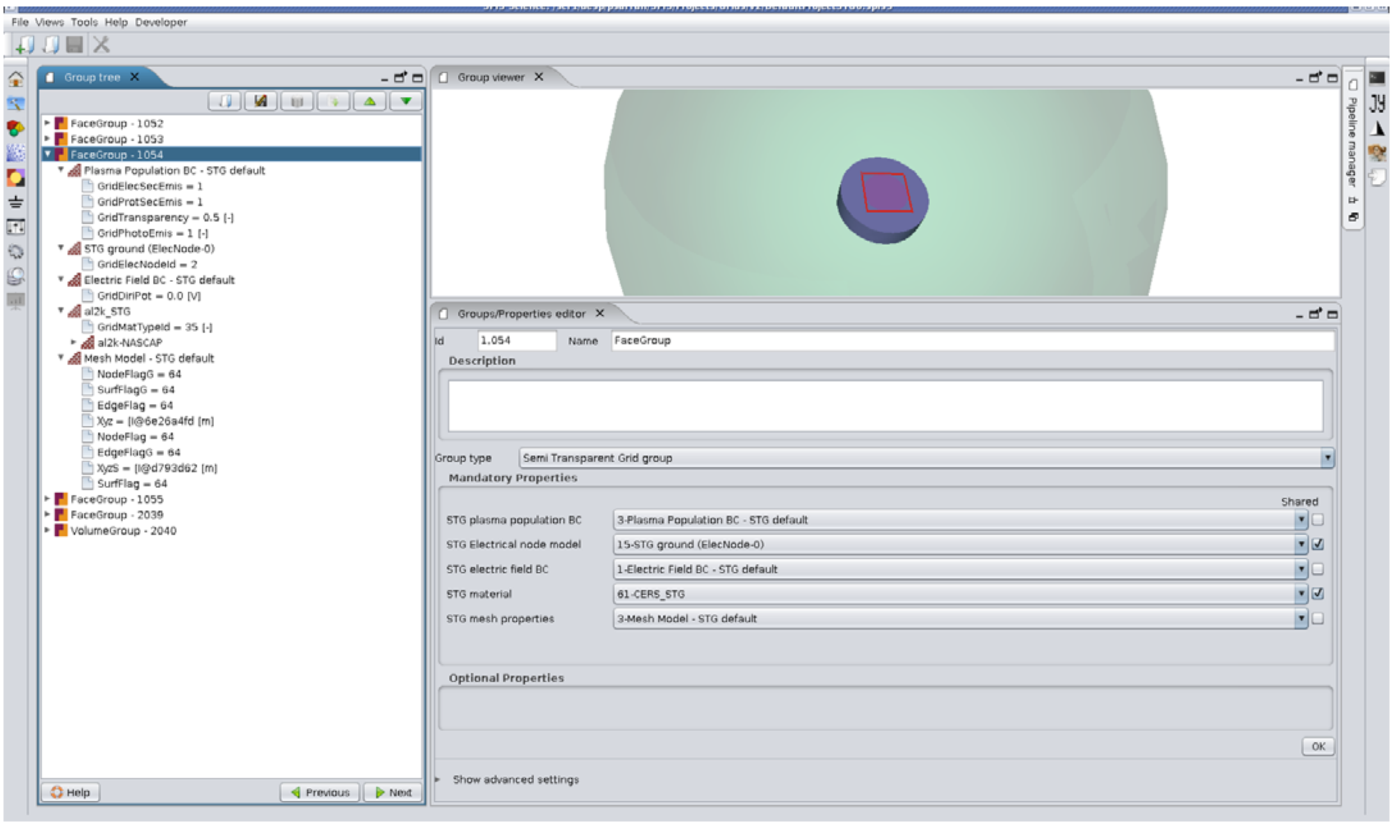


Figure 28 - Design of the surface meshes associated to a STG

3.3.2. *Inputs*



```
FaceGroup - 1052
FaceGroup - 1053
FaceGroup - 1054
  Plasma Population BC - STG default
    GridElecSecEmis = 1
    GridProtSecEmis = 1
    GridTransparency = 0.5 [-]
    GridPhotoEmis = 1 [-]
  STG ground (ElecNode-0)
    GridElecNodeId = 2
  Electric Field BC - STG default
    GridDiriPot = 0.0 [V]
  al2k_STG
    GridMatTypeId = 35 [-]
    al2k-NASCAP
  Mesh Model - STG default
    NodeFlagG = 64
    SurfFlagG = 64
    EdgeFlag = 64
    Xyz = [I@6e26a4fd [m]
    NodeFlag = 64
    EdgeFlagG = 64
    XyzS = [I@d793d62 [m]
    SurfFlag = 64
FaceGroup - 1055
FaceGroup - 2039
VolumeGroup - 2040
```

Groups/Properties editor

Id: 1.054 Name: FaceGroup

Description

Group type: Semi Transparent Grid group

Mandatory Properties

		Shared
STG plasma population BC	3-Plasma Population BC - STG default	☐
STG Electrical node model	15-STG ground (ElecNode-0)	✓
STG electric field BC	1-Electric Field BC - STG default	☐
STG material	61-CERS_STG	✓
STG mesh properties	3-Mesh Model - STG default	☐

Optional Properties

Figure 29 – STG definition in the group editor

For each STG, the user should select in the Group editor:
- The grid transparency
- The electric super node linked to the grid
- The grid potential if the grid is not linked to the SC circuit
- The material of the grid (only metal permitted)
- The mesh model of STG (only one model available)

There are no specific global parameters for semi-transparent grids. All the GP defined for spacecraft interaction with the plasma also apply to STGs.

3.3.3. *Particle pusher*

The grid transparency is taken into account by all particle volume distributions (PIC, Backtracking and Composite). A particle reaching the grid has a probability to pass through the STG corresponding to the transparency. The transparency is treated statistically. When a particle is collected, the particle is added to the surface distribution function living on the STG. When a particle is not collected, the particle continues its trajectory without any modification.

SPIS solvers make the difference between the particles collected on the face A and on the face B. It ensures a correct computation of secondary populations on the both sides of the STG (especially in the case of photoemission where the collected population is photon and the emission is present only on one face of the grid).
NB: The transparency is also taken into account for photons: transparency shading effect of photoemission is modelled.

The emission of secondary populations is activated with the same flag as on the SC. A STG can support: photoemission, secondary emission by electron impact, secondary emission by proton impact and hoping effect.

3.3.4. Results monitoring

The visualization of the results is performed on both sides of a STG. The 3D outputs (i.e. color mapping) being done MeshGroup by MeshGroup, it is possible to visualize on both or only one side of the STG by selection of the relevant MeshGroup. Because a mesh is defined for each STG, each STG can be visualize independently. In both cases, the selection of the data displayed is done by selection of the relevant DataField in the Data Miner.

3.3.5. Graphical User Interface

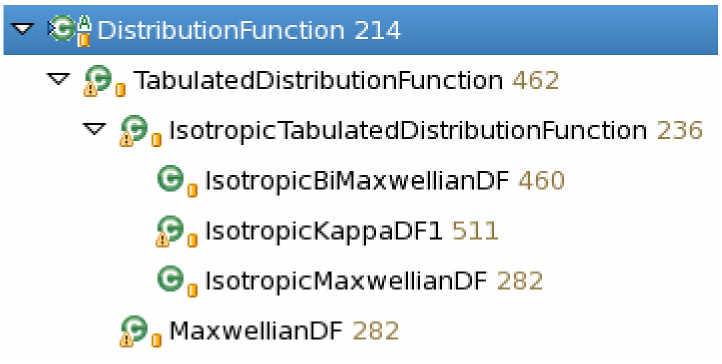
The User Interfaces elements dedicated to STG are similar to the Instruments ones. The geometry of STG is defined through the CAD tool (i.e Gmsh). The surface of the STG should be defined as a surface limiting two volumes constituting the computational volume, like 2D thin elements. As for 2D thin elements, a MeshGroup is generated for each face.

The responsibility of STG geometry consistency and of the respective attribution of faces (A and B) is let to the user. The designation of each face is done by selection of the respective MeshGroup through the Groups Manager.

The generation of the additional meshes will be done by the extraction of the corresponding surface mesh elements (as indicated by the STG flag) using the features of the Penelope library.

3.4. Generic distribution functions

A new set of velocity distribution functions has been developed to increase the code capabilities, which previously supported only Maxwellian distributions for ambient particles (possibly drifting for ions). Distribution functions (DF) of PIC populations are passed or transformed internally in tabulated DF. The most general case is a DF defined versus three velocity components, basically in a Cartesian coordinates system, but also possibly in polar coordinates.


DistributionFunction 214
TabulatedDistributionFunction 462
IsotropicTabulatedDistributionFunction 236
IsotropicBiMaxwellianDF 460
IsotropicKappaDF1 511
IsotropicMaxwellianDF 282
MaxwellianDF 282

All the distribution functions extending DistributionFunction are possible to use when invoking a PICVolDistrib population, simply by setting its name in the global parameter *popNEnvironmentDF*. In the following table, the list of global parameters needed to define an extended population is represented. This list contains an adaptation of all the parameters needed to buid a biMaxwellianEnvironment as the density for exemple popNDensity. But at the difference of the biMaxwellianEnvironment population, the secondary emission flag is externalized pop1SEEFlag.

Some additional parameters are also needed depending on the class of the distribution function. For example, a file path has to be defined in the case of an IsotropicTabulatedDistributionFunction.

Name	Description	Variab le type	Unit	Default value	In use
pop1Distrib	Name od the distribution function	String	[-]	PICVolDistrib	yes
pop1Type	Particle type of the population	String	[-]	None	yes
pop1Density	Density of particles at the infinity	Float	[m-3]	All	yes
pop1Dt	Integration time step	float	[s]	0,0	yes
pop1Duration	Integration duration	float	[s]	0,0	Yes
pop1TrajFlag	Trajectory flag	Int	[-]	0	yes
pop1SpeedUp	Population speed up	Float	[-]	0	yes
pop1EnvironmentDF	Name of the distribution function to be injected	String	[-]	TabulatedDistributionFunction	yes
pop1DF_FileName	File path in case of a tabulated DF	String	[-]	test1DFmode1_exemple1.txt	yes
pop1SEEFlag	Secondary electron emission flag by electron impact	Int	[-]	0	yes

3.4.1. Generic user defined distribution function

The user defined distribution functions are concerned by two new classes. The user defined distributions are defined by tabulated files. Three different file formats are available: two of them concern the definition of a full 3V distribution function without constraints; the last one permits to define an isotropic DF.

Mode 1 permits to define the DF in a rectangular and structured mesh. In the following example, the user first defines the three scales (X, Y, Z) corresponding to (vx, vy, vz). Then, the value of the DF is defined by a 2D matrix with values corresponding to each step in the X scale.

```
→ TabulatedDistributionFunction (mode 1)
# X tab (table of X values)
-200000.0
-100000.0
0.0
200000.0
400000.0
# Y tab (table of Y values)
-5.0e+4
-2.5e+4
-1.0e+4
0.0
1.0e+4
1.5e+5
3.0e+5
# Z tab (table of Z values)
-8e+4
-4e+4
0.0
1.0e+5
1.5e+5
# X = -2000.0
0.0         0.0         0.0         0.0         0.0
0.0         0.0         0.0         0.0         0.0
0.0         0.0         0.0         0.0         0.0
0.0         0.0         0.0         0.0         0.0
0.0         0.0         0.0         0.0         0.0
0.0         0.0         0.0         0.0         0.0
0.0         0.0         0.0         0.0         0.0
# X = -1000.0
0.0         0.0         0.0         0.0         0.0
0.0         0.9         0.9         0.9         0.0
0.0         0.9         0.9         0.9         0.0
0.0         0.9         0.9         0.9         0.0
0.0         0.9         0.9         0.9         0.0
0.0         0.9         0.9         0.9         0.0
0.0         0.0         0.0         0.0         0.0
# X = 0.0
…
```

The most general way to define a 3V distribution function is however adopted in mode 2, see next table. The DF is described in an unstructured mesh of the phase space. Each element of the mesh is a slab with origin (vx, vy, vz) and dimensions (dvx, dvy, dvz). In each element, the distribution is trilinear. The user must define the value *f* at the origin and the slopes of *f* (df/dvx, df/dvy, df/dvz) within the slab. This second mode is more general but the meshing is let to the user's responsibility. It can be noted that where the DF is not defined, its value is zero (no extrapolation performed).

```
→ IsotropicTabulatedDistributionFunction  (mode  2)
# vx        vy          vy          f           dvx         dvy         dvz         df/dvx      df/dvy
            df/dvz
-100.0      -10.0       -10.0       1.0         90.0        10.0        10.0        0.0         0.0         0.0
-100.0      -10.0       10.0        1.0         90.0        10.0        -10.0       0.0         0.0         0.0
-100.0      10.0        -10.0       1.0         90.0        -10.0       10.0        0.0         0.0         0.0
-100.0      10.0        10.0        1.0         90.0        -10.0       -10.0       0.0         0.0         0.0
-10.0       -10.0       -10.0       1.0         10.0        10.0        10.0        0.0         0.0         0.0
-10.0       -10.0       10.0        1.0         10.0        10.0        -10.0       0.0         0.0         0.0
-10.0       10.0        -10.0       1.0         10.0        -10.0       10.0        0.0         0.0         0.0
-10.0       10.0        10.0        1.0         10.0        -10.0       -10.0       0.0         0.0         0.0
0.0         -10.0       -10.0       1.0         2.0         10.0        10.0        0.0         0.0         0.0
0.0         -10.0       10.0        1.0         2.0         10.0        -10.0       0.0         0.0         0.0
0.0         10.0        -10.0       1.0         2.0         -10.0       10.0        0.0         0.0         0.0
0.0         10.0        10.0        1.0         2.0         -10.0       -10.0       0.0         0.0         0.0
2.0         -10.0       -10.0       1.0         10.0        10.0        10.0        0.0         0.0         0.0
2.0         -10.0       10.0        1.0         10.0        10.0        -10.0       0.0         0.0         0.0
2.0         10.0        -10.0       1.0         10.0        -10.0       10.0        0.0         0.0         0.0
2.0         10.0        10.0        1.0         10.0        -10.0       -10.0       0.0         0.0         0.0
…
```

Six global parameters define the basis on which the DF is calculated:

Name	Description	Variab le type	Unit	Default value	In use
pop1DFBasis_ Vect1_X	X component of the vect1 defining the DF base	Float	[-]	1.0	yes
pop1DFBasis_ Vect1_Y	Y component of the vect1 defining the DF base	Float	[-]	1.0	yes
pop1DFBasis_ Vect1_Z	Z component of the vect1 defining the DF base	Float	[-]	1.0	yes
pop1DFBasis_ Vect2_X	X component of the vect2 defining the DF base	Float	[-]	1.0	yes
pop1DFBasis_ Vect2_Y	Y component of the vect2 defining the DF base	Float	[-]	1.0	Yes
pop1DFBasis_ Vect2_Z	Z component of the vect2 defining the DF base	Float	[-]	1.0	yes

NB: After the importation of a user-defined tabulated file, a renormalisation of the distribution is performed, in order to insure that the population density imposed by the user in the global parameters will be really imposed in undisturbed plasma regions:

$$f\left(\mathrm{v}_x,\mathrm{v}_y,\mathrm{v}_z\right)=\frac{f_{from\_file}\left(\mathrm{v}_x,\mathrm{v}_y,\mathrm{v}_z\right)}{\iiint\limits_{\mathrm{v}\in[-\infty,+\infty]} f_{from\_file}\left(\mathrm{v}_x,\mathrm{v}_y,\mathrm{v}_z\right)d\mathrm{v}_x d\mathrm{v}_y d\mathrm{v}_z}$$

This class ant its extensions provide sampling methods for velocity. Three different methods of sampling are implemented:

- An acceptance rejection method
- A numerical inversion method
- A direct weight mode

The aim of all these methods is to provide a set of (vx,vy,vz,w) following a user defined distribution function. The most efficient in the case of a tabulated function is the numerical inversion method. It is based on the fact that the distribution function is already discretized in the phase space. For each phase space element *i*, located at coordinates (vx,vy,vz), we have:

$$f_i = f\left(\mathrm{v_x}, \mathrm{v_y}, \mathrm{v_z}\right)$$

$$d\mathrm{v}_i = d\mathrm{v_x}\mathrm{dv_y}\mathrm{dv_z}$$

A mesh element *n* of the phase space is then randomly chosen following the relation:

$$\sum_{i=0}^{n} f_i d\mathrm{v}_i = R$$

with R a random number.

In the cell n, the final velocity of the particle is randomly sampled between (vx,vy,vz) and (vx+dvx,vy+dvy,vz+dvz). The weight of the particle is uniform in on cell of the space meshing. It is deduced from the number of particle in the cell and the density in this cell. If the sampling is done in surface, the same process is used with a surface distribution function:

$$f_{surf}\left(\vec{\mathrm{v}}\right) = \left(\vec{\mathrm{v}} \bullet \vec{\mathrm{n}}\right) f_{surf}\left(\vec{\mathrm{v}}\right)$$

3.4.2. Isotropic user defined distribution function

The IsotropicTabulatedDistributionFunction extends the TabulatedDistributionFunction class. It is not needed to provide a 3V tabulated distribution function. The user must only tabulate the distribution function as a function of the norm of the distribution function fn(v).

$$f\left(\mathrm{v_x}, \mathrm{v_y}, \mathrm{v_z}\right)\mathrm{dv_x}\,\mathrm{dv_y}\,\mathrm{dv_z} = f_n\left(\mathrm{v}\right)\mathrm{v}^2\ \cos\left(\theta\right)\mathrm{dv}\,\mathrm{d}\theta\,\mathrm{d}\phi$$

A linear interpolation is done between each point of the tabulated distribution function of the velocity norm.

v	*f(v)*
1.000000E+01	*5.210000E-01*
1.200000E+01	*5.210000E-01*
1.440000E+01	*5.210000E-01*
1.728000E+01	*5.210000E-01*
2.073600E+01	*5.210000E-01*
2.488320E+01	*5.210000E-01*
2.985984E+01	*5.210000E-01*
3.583181E+01	*5.210000E-01*
4.299817E+01	*5.210000E-01*
5.159780E+01	*5.210000E-01*
6.191736E+01	*5.210000E-01*
7.430084E+01	*5.210000E-01*
8.916100E+01	*5.210000E-01*
1.069932E+02	*5.210000E-01*
1.283918E+02	*5.210000E-01*
1.540702E+02	*5.210000E-01*
1.848843E+02	*5.210000E-01*
2.218611E+02	*5.210000E-01*
2.662333E+02	*5.210000E-01*
…	

3.4.3. Bi-Maxwellian distribution function

This extended population is selected through the GP pop#EnvironmentDF parameters for environment plasma populations (and by the corresponding GP for photoelectrons) by typing directly "IsotropicBiMaxwellianDF"

Name	Description	Variable type	Unit	Default value	In use
pop1Temperature1	Population first temperature	Float	[eV]	1.0	yes
pop1Temperature2	Population second temperature	Float	[eV]	1.0	yes
pop1RatioN1overN2	Ratio between the density of the bi-maxwellian population	Float	[-]	1.0	yes (if pop1RatioJ1overJ2 not set)
pop1RatioJ1overJ2	Ratio between the current density of the bi-maxwellian population	Float	[-]	1.0	yes

3.4.4. Kappa distribution function

An isotropic Kappa distribution function has been implemented following:

$$f_\kappa(v) = \frac{N}{\pi^{3/2}} \frac{1}{\theta^3} \frac{\Gamma(\kappa+1)}{\kappa^{3/2}\Gamma(\kappa-1/2)} \left(1+\frac{v^2}{\kappa\theta^2}\right)^{-(\kappa+1)}$$

$$\theta = \left[(2\kappa-3)/\kappa\right]^{1/2} (kT/m)^{1/2}$$

The IsotropicKappaDF1 extends the IsotropicTabulatedDistributionFunction. An automatic tabulation of this Kappa function is performed. The methods of the parent class IsotropicTabulatedDistributionFunction permit to sample the particles. For backtracking methods, the returned value of the DF comes from the analytic equation and not from the tabulated ones.

3.4.5. Application of a drift velocity

Adding a drifting velocity is possible for all user-defined distribution functions. This function of the TabulatedDistributionFunction class makes a drift in the distribution function without performing again the DF tabulation.

$$f(\mathrm{v}_x, \mathrm{v}_y, \mathrm{v}_z) = f_0(\mathrm{v}_x - \mathrm{v}_{x,\mathrm{drift}}, \mathrm{v}_y - \mathrm{v}_{y,\mathrm{drift}}, \mathrm{v}_z - \mathrm{v}_{z,\mathrm{drift}})$$

3.5. Surface interactions update

3.5.1. Extended material properties

We remind that the *Properties and Groups Editor* of SPIS is based on the Frida library. In this one, Properties (e.g. material) are structured in a tree and hierarchized structure. Each Property gather a set of Characteristics, these ones being of various types like scalars (i.e. float, double or int), Strings, objects or series double values. In Series Characteristics, several series of values can be defined.

Series Characteristics allows SPIS to attribute rich properties with tabulated values (e.g. tabulated secondary emission function versus energy) to groups. Example of rich Properties and tabulated are provided with tests projects of the present study. To create new Properties, several approaches are possible. New properties files can be edited manually as all XML file. However, the simplest one is to use the advanced editor in the selection tree *Properties and Groups Editor* as follow:

1. Select the group to which one the Property should be attributed to;
2. On the selected group, access to the contextual menu with a right-button click, add a new Property.
3. Select the newly created property and add, in the same way, a new Characteristics of type Series, as illustrated in Figure 30.

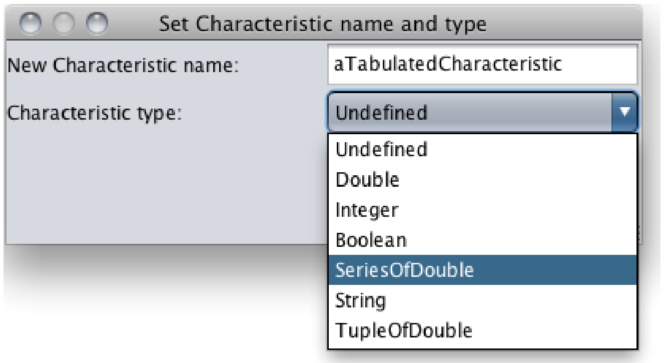
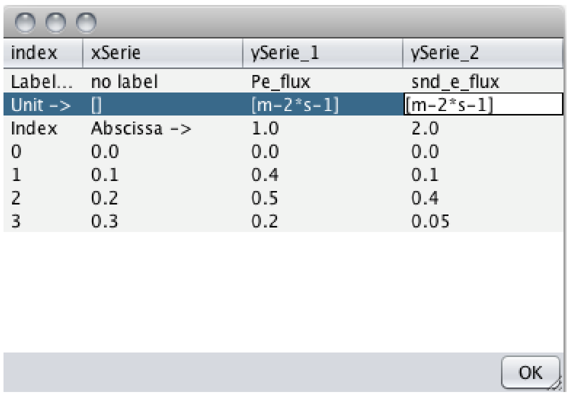


Figure 30: Creation of a new Characteristic of type series of double.

4. Set the number of series (i.e. number of columns) and the size of the series (i.e. number of values by series);
5. Click in the Edit button to set the values of series. An editor should appear as illustrated in Figure 31.

UNCLASSIFIED



index	xSerie	ySerie_1	ySerie_2
Label...	no label	Pe_flux	snd_e_flux
Unit ->	[]	[m-2*s-1]	[m-2*s-1]
Index	Abscissa ->	1.0	2.0
0	0.0	0.0	0.0
1	0.1	0.4	0.1
2	0.2	0.5	0.4
3	0.3	0.2	0.05

Figure 31: Editor of Series Characteristics. Here two series of four values each are defined. Labels and units can be defined for each series.

6. By clicking on the plot button, series can be displayed as illustrated in Figure 32.

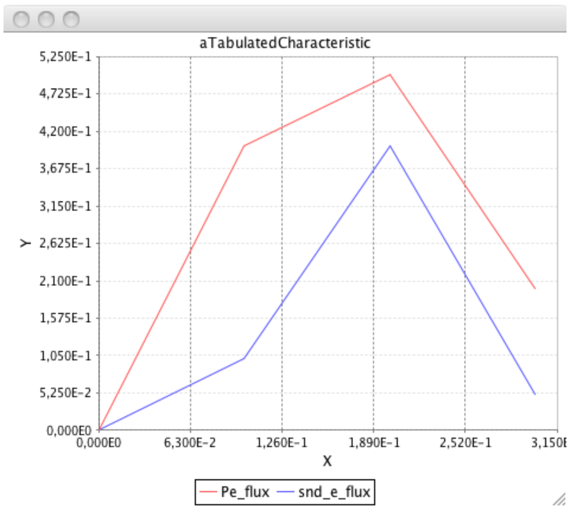


Figure 32: Example of tabulated characteristics.

7. After a complete setting of the Property and its related Characteristics, select the built Properties, with the right-click button, access to the contextual module and select the “Save this property” item, as illustrated in Figure 33.

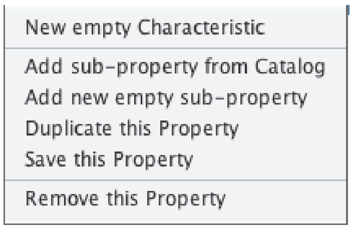


Figure 33: Contextual menu to save the selected Property.

We recommend to same all selected Property into a directory with an “.xcat” extension (should be created manually before) to constitute a new catalogue. Such directory can then be loaded as usual.

UNCLASSIFIED

```
<Property>
  <id>99</id>
  <name>exampleOfRichProperty</name>
  <description>another property</description>
  <characteristicList>
    <entry>
      <string>aTabulatedCharacteristic</string>
      <SeriesOfDoubleCharacteristic>
        <id>-1</id>
        <name>aTabulatedCharacteristic</name>
        <parentProperty class="Property" reference="../../../.."/>
        <unit class="SimpleStringUnit">
          <unit>[m]</unit>
          <standardUnit>m</standardUnit>
        </unit>
        <localisation>-1</localisation>
        <informationLink></informationLink>
        <value>
          <double-array>
            <double>0.0</double>
            <double>0.1</double>
            <double>0.2</double>
            <double>0.3</double>
          </double-array>
          <double-array>
            <double>0.0</double>
            <double>0.4</double>
            <double>0.5</double>
            <double>0.2</double>
          </double-array>
          <double-array>
            <double>0.0</double>
            <double>0.1</double>
            <double>0.4</double>
            <double>0.05</double>
          </double-array>
        </value>
        <xSerie reference="../value/double-array"/>
        <seriesAbscissa>
          <null/>
          <double>1.0</double>
          <double>2.0</double>
        </seriesAbscissa>
        <ySeriesLabel>
          <string>no label</string>
          <string>Pe_flux</string>
          <string>snd_e_flux</string>
        </ySeriesLabel>
        <seriesUnit>
          <SimpleStringUnit>
            <unit>[]</unit>
            <standardUnit></standardUnit>
          </SimpleStringUnit>
          <SimpleStringUnit>
            <unit>[m-2*s-1]</unit>
            <standardUnit>m-2*s-1</standardUnit>
          </SimpleStringUnit>
          <SimpleStringUnit>
            <unit>[m-2*s-1]</unit>
            <standardUnit>m-2*s-1</standardUnit>
          </SimpleStringUnit>
        </seriesUnit>
        <size>4</size>
      </SeriesOfDoubleCharacteristic>
    </entry>
  </characteristicList>
  <isCompound>false</isCompound>
  <type></type>
</Property>
```

Table 3 - Example of saved rich Property with Series Characteristics.

3.5.2. Generic secondary populations

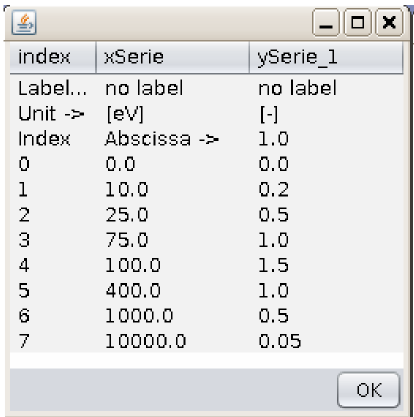
Three new classes have been developed in order to extend the possibilities for secondaries:

- For photoemission: the GenericPhotoEmInteractor extends the PhotoEmInteractor by permitting to have generic distribution function instead of Maxwellian.
- For secondary emission by proton impact: the GenericSEEPInteractor extends the BasicSEEPInteractor by permitting to have generic distribution function instead of Maxwellian for emission.
- For secondary emission by electron impact: the GenericSEEEInteractor extends the BasicSEEEInteractor by permitting to have generic distribution function instead of Maxwellian for the true emission and a tabulated yield for true SEE and backscattered.

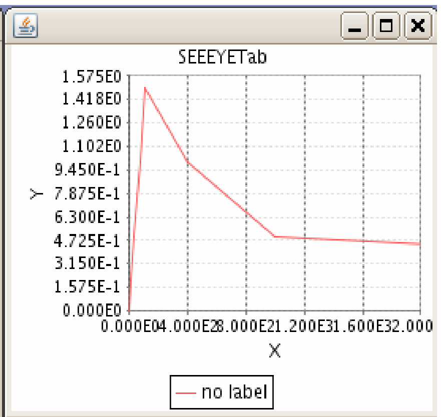
3.5.2.1. User defined yields

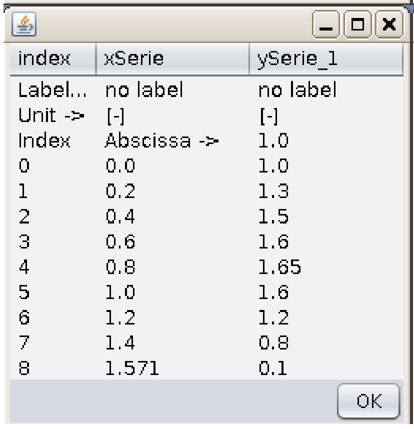
SEEE yield can also be tuned with extended material properties of SPIS-5, through tabulated data. The user can add new material properties to the material properties list, at group editor level. The flag parameters permit to activate the tabulated yields instead of the legacy analytical models.

Name	**Description**	**Variable type**	**Unit**	**Default value**
SEEEYFlag	SEE by electron yield dependence in energy flag (0-analytical model / 1-tabulated mode)	Int	[-]	0.0
SEEEYETab	SEE by electron dependence in energy tab	Array	[eV,-]	-
SEEEYTTab	SEE by electron dependence in angle tab	Array	[rad,-]	-
BCKEAFlag	Backscattering of electron albedo dependence in energy flag (0-analytical model / 1-tabulated mode)	Int	[-]	0.0
BCKEAETab	Backscattering of electron albedo dependence in energy tab	Array	[eV,-]	-
BCKEATTab	Backscattering of electron albedo dependence in angle tab	Array	[rad,-]	-



index	xSerie	ySerie_1
Label...	no label	no label
Unit ->	[eV]	[-]
Index	Abscissa ->	1.0
0	0.0	0.0
1	10.0	0.2
2	25.0	0.5
3	75.0	1.0
4	100.0	1.5
5	400.0	1.0
6	1000.0	0.5
7	10000.0	0.05





index	xSerie	ySerie_1
Label...	no label	no label
Unit ->	[-]	[-]
Index	Abscissa ->	1.0
0	0.0	1.0
1	0.2	1.3
2	0.4	1.5
3	0.6	1.6
4	0.8	1.65
5	1.0	1.6
6	1.2	1.2
7	1.4	0.8
8	1.571	0.1

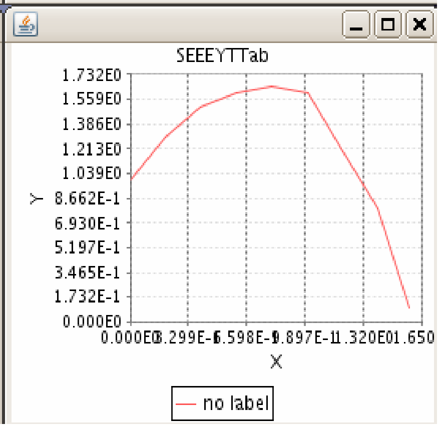


Figure 34 -Example of tabulated SEEE yield as a function of impacting electron energy (absolute value) and incidence angle (relative value)

3.5.2.2. User defined distribution function

Secondary electrons from electron, proton and photon impact can be modelled with the isotropic distribution functions defined in the previous section (no warrantee for 3V DF since the reference basis is applied whatever the surface of injection). It behaves as extended populations for environment, the same global parameters format is used.

3.5.3. Self-shading

This development was performed in the frame of the SPIS-GEO contract. However, for the sake of completeness, we present here its design.

The photoemission process has been enhanced taking account of the spacecraft self-shading, i.e. surfaces oriented towards the Sun but masked by other spacecraft surfaces. The numerical method is a *ray tracing* technique (particles emitted from spacecraft and straight trajectory). The initialization of the photoemission current backtracks a uniform velocity population, emitted in the direction of the Sun (see *initPhotoemission*() of *simulationFromUIParams*). When a particle reaches the external boundary, it contributes to the current emitted on the spacecraft surface element of origin. Otherwise, i.e. when intercepted by other spacecraft elements, the particle is discarded. On the example of Figure 35, the voluntarily large cylindrical antenna shades the spacecraft resulting in a reduced current density.

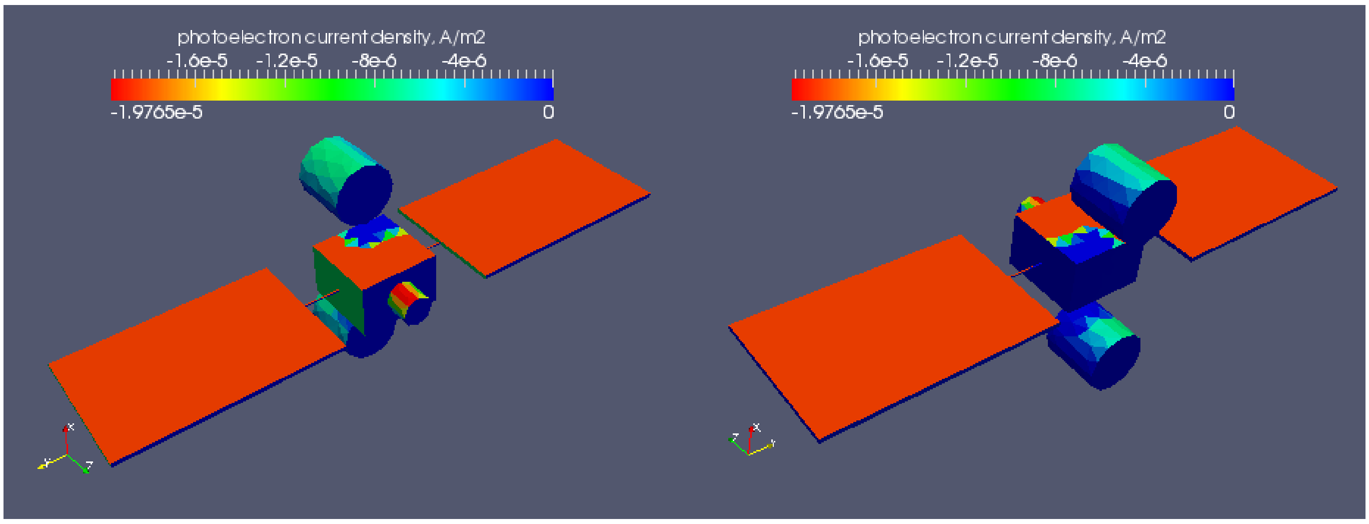

photoelectron current density, A/m2
-1.6e-5
-1.2e-5
-8e-6
-4e-6
-1.9765e-5
0
photoelectron current density, A/m2
-1.6e-5
-1.2e-5
-8e-6
-4e-6
-1.9765e-5
0

Figure 35 - Illustrations of spacecraft self-shading leading to a limitation or cancellation of the photoemission process on partially or fully masked surfaces

3.5.4. Solar array interactor

The development of "Solar arrays plasma interaction (SR-FGS-002)" of SPIS-SCI concerns the effect of small metallic units set to high voltages, i.e. interconnectors of a solar array. It aims at modelling them not at the length scale of the panel and not at the length scale of the interconnectors (not meshed). At this scale, we can make the assumption that the solar array surfaces are dielectric surfaces through which a part of current is collected directly by the ground, i.e. not by the dielectric surfaces of the solar cell. The electric field is also disturbed at the vicinity of these highly biased small elements.

The interconnector modelling is based on the next assumptions:

- The solar array mesh need not being adapted to the interconnects size
- The current collection on the solar array is calculated as a standard simulation, i.e. the plasma is not globally affected by the interconnect
- Locally in each cell of the solar array, the current is distributed between the cover-glasses and the interconnects
- The current distribution is affected by the potential of the interconnects behind the cover-glasses (map defined by the user).

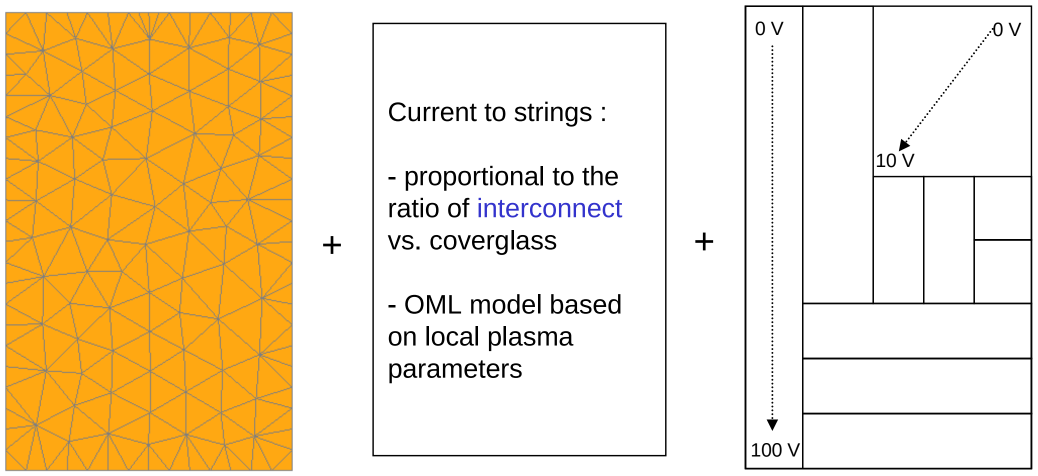

+
Current to strings :
- proportional to the ratio of interconnect vs. coverglass
- OML model based on local plasma parameters
+
0 V
0 V
10 V
100 V

Figure 36- Schematic view of the interconnector modelling

3.5.4.1. How to define an interconnect in SPIS?

The SolarArrayInteractor must be defined in:

- GlobalParams:
  - interactorNb = 1
  - interactorParticleType1 = None
  - interactorPopSource1 = All
  - interactorType1 = SolarArrayInteractor
- LocalParams: Plasma model type spacecraft
  - SourceId = 1001 (i.e. 1000 + InteractorId)

The description files must be created in Project -> NumKernel -> Input

- Interactor1_SolarArrayDescriptor.txt gives the potential map on the solar Array.
- Interactor1_SolarArrayCollectionRation.txt gives the current ratio collected by the interconnect as a function of the relative reduce potential of the interconnect. This file is optional. If it is not present an OML like law is used instead.

3.5.4.2. Solar array potential map

The solar array potential map and geometry is defined in the first file named "Interactor1_SolarArrayDescriptor.txt" for the Interactor1. This file is defined as follow:

- The first value corresponds to the geometric ratio of the interconnect (surface of interconnect/total surface of the panel) -> range [0:1]

NB: this ratio is only used with the OML approximation => not used if Interactor1_SolarArrayCollectionRation.txt exists

- Basis definition:

  xM yM zM (coordinate of the M point in the SPIS geom ref)
  xX yX zX (MX vector definition)
  xY yY zY (MY vector definition)

Panel elements are built in the (M,MX,MY) base

- Definition of one rectangular zone of the panel by lines (see Figure 2):

  x0 y0 x1 y1 V0 dV/dx dV/dy

  For example for the first zone:

  0.0 0.0 4.0 0.2 0.0 25.0 0.0

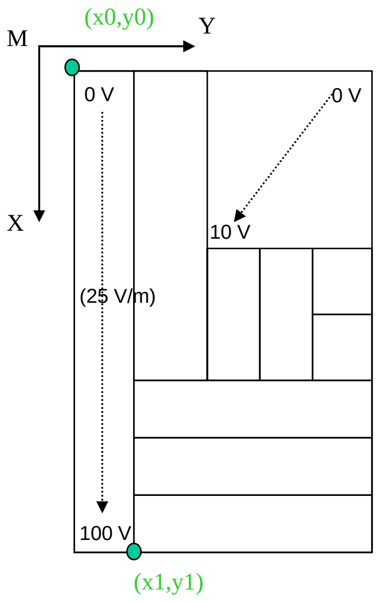


Figure 37 - Definition of a solar array potential map by zones where the potential evolves linearly

3.5.4.3. Solar array collection law

The ratio of current collection is defined as a function of the reduced differential potential and a geometrical factor. The differential potential is defined as the difference between the potential at the dielectric surface and the potential of the electrical node of the solar array. Then, the reduced differential potential is the ratio between the differential potential and the particles energy in [eV]. When the current ratio is defined in a file, both effects are supposed to be taken into account within the file. The user should create a file named “Interactor1_SolarArrayCollectionRation.txt” where the probability for a particle with a known energy is set as a function of the reduced potential.

File example:

```
-1000.0      0.0
-10.0        0.0005
-1.0         0.01
0.0          0.01
1.0          0.01
10.0         0.1 (i.e: if the diff pot is 10V, the probability for the interco to collect an electron at
1eV is 10%)
100.0        0.2
1000.0       0.5
```

When no file is defined, a cylindrical OML like law is used. The probability for a particle to be collected is:

$$proba = \frac{R \times \sqrt{1 + V_{diff,reduced}}}{(1-R) + R \times \sqrt{1 + V_{diff,reduced}}}$$

Where R is the geometric ratio (defined in Interactor1_SolarArrayDescriptor.txt) and V_{diff} is the reduced differential potential.

3.6. Thin wires

The main development concerning thin wires is was the current collection and the current emission from these elements. These developments impacted a lot of existing classes concerning:

- the particle pusher
- the surface distribution function
- the surface interactors
- the spacecraft circuit

3.6.1. Particle pusher

The particle pusher has been modified to collect particles on a wire. A wire is represented by an unmeshed cylinder around an edge of the mesh. In this case, the standard pusher (exact when E is uniform) is not used. The *crossTetraUniformEwithDichotomy* permits to take into account the potential singularity around the wire (see magnetic field section).

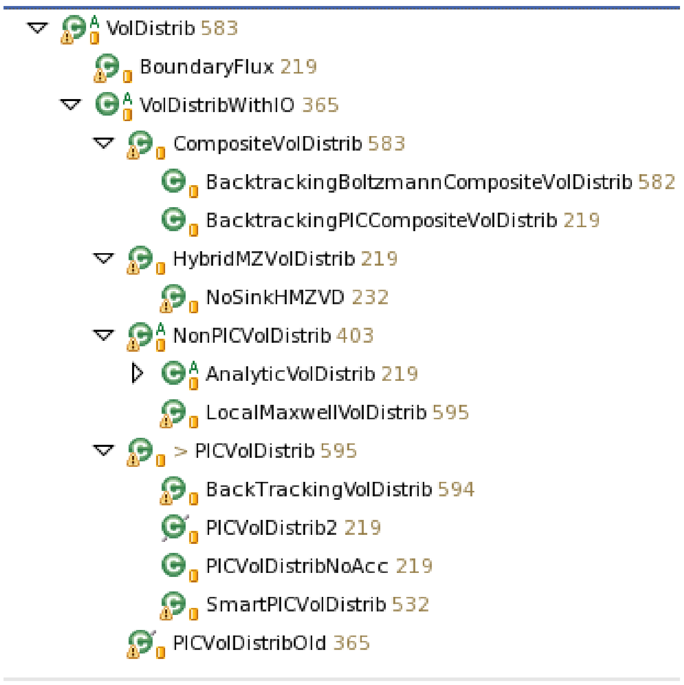
3.6.2. Surface distribution

A major development concerns the fact that the surface distributions in the previous version of SPIS were all located on surface elements. Now due to thin wires extension, a surface distribution can live both on standard surface elements but also on edges defined as wires within the user interface.

The development has consisted on defining a MultipleSurfDistrib class instead of SurfDistrib for all collected and emitted currents. The MultipleSurfDistrib is composed of two SurfDistrib, by convention:

- The first is located on the SC surfaces → surface elements of the SC mesh
- The second is located on the SC wires → edge elements of the SC mesh

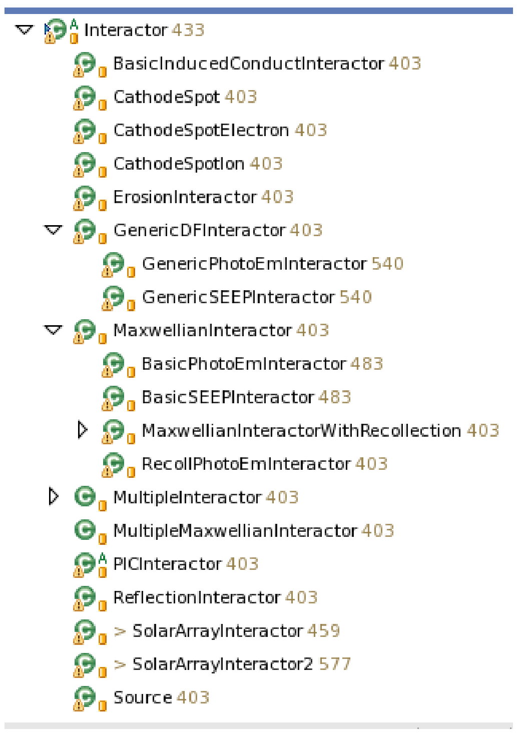
This modification impact all VolDistrib classes and extensions (full list below). The source terms on spacecraft surfaces are also compatible with MultipleSurfDistribs (needed for photo-electrons emission and etc…). All the class working with the input flux or calculating the output flux of a VolDistrib have been made compatible with MultipleSurfDistrib.



```
▽ VolDistrib 583
      BoundaryFlux 219
   ▽ VolDistribWithIO 365
      ▽ CompositeVolDistrib 583
            BacktrackingBoltzmannCompositeVolDistrib 582
            BacktrackingPICCompositeVolDistrib 219
      ▽ HybridMZVolDistrib 219
            NoSinkHMZVD 232
      ▽ NonPICVolDistrib 403
         ▷ AnalyticVolDistrib 219
            LocalMaxwellVolDistrib 595
      ▽ > PICVolDistrib 595
            BackTrackingVolDistrib 594
            PICVolDistrib2 219
            PICVolDistribNoAcc 219
            SmartPICVolDistrib 532
         PICVolDistribOld 365
```

3.6.3. *Surface interaction*

The surface interactors have also been made compatible with thin wires, by using MultipleSurfDistrib as input and output (full list of interactors below).



```
▽ Interactor 433
    BasicInducedConductInteractor 403
    CathodeSpot 403
    CathodeSpotElectron 403
    CathodeSpotIon 403
    ErosionInteractor 403
  ▽ GenericDFInteractor 403
      GenericPhotoEmInteractor 540
      GenericSEEPInteractor 540
  ▽ MaxwellianInteractor 403
      BasicPhotoEmInteractor 483
      BasicSEEPInteractor 483
    ▷ MaxwellianInteractorWithRecollection 403
      RecollPhotoEmInteractor 403
  ▷ MultipleInteractor 403
    MultipleMaxwellianInteractor 403
    PICInteractor 403
    ReflectionInteractor 403
    > SolarArrayInteractor 459
    > SolarArrayInteractor2 577
    Source 403
```

Concerning the secondary emission by proton and electron impact, a particular attention has been paid to take account of the incidence angle of impacting populations on the wire for the calculation of the yield. Knowing the position of the particle on the wire at the impact and knowing the wire position, the exact incidence angle is calculated in SPIS.

3.6.4. Spacecraft circuit

In the Spacecraft circuit, the current collected on wires and emitted from wires is passed using different SurfField from the SurfField used for the currents on the SC. The current collected by the wires is then mapped onto the SC circuit nodes elements and the circuit solver is used as usual with current on electrical nodes (i.e. CircFields).

As an output of the circuit solver, the potential on the electrical super nodes representing is mapped to the edges of the computational volume mesh.

3.6.5. Monitoring

The monitors of surface fields have been updated in order to plot the wire data on the edges.

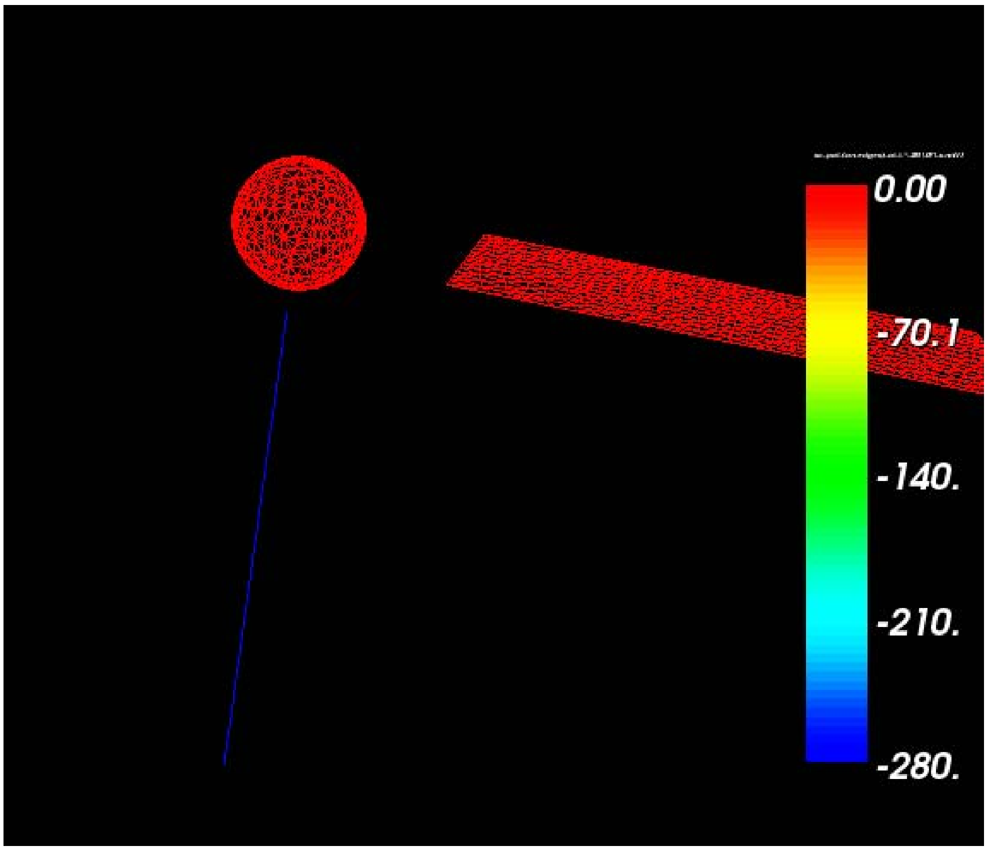


Figure 38 – Plot of the potential on the wire on the SC

3.7. Pre-defined transitions

3.7.1. Objectives

An important aspect of the new SPIS capabilities is the modification of parameters within the course of the simulation. The methodology described in this section is adapted to the transitions/scenarios to be implemented in the frame of this activity (spinning spacecraft, plasma source activation, potential sweep between electrical nodes) and in the frame of SPIS-GEO [AD2]-[AD3] (eclipse exit) but can be easily extrapolated to lots of other situations.

3.7.2. General design

The logical diagram of the interaction of the simulation with transitions is presented in Figure 39. From the user point of view, transitions are defined during the simulation global parameters setting and are run automatically, i.e. without any further interaction, during the simulation integration. A transition is able to pause the simulation, change a variety of its parameters and resume the simulation, using a series of so-called *Updater*. *Updaters* actually change simulation parameters while *transitions* control the value of the parameters versus time to be applied to *updaters*. Doing this, different transitions can access to the same type of updaters.

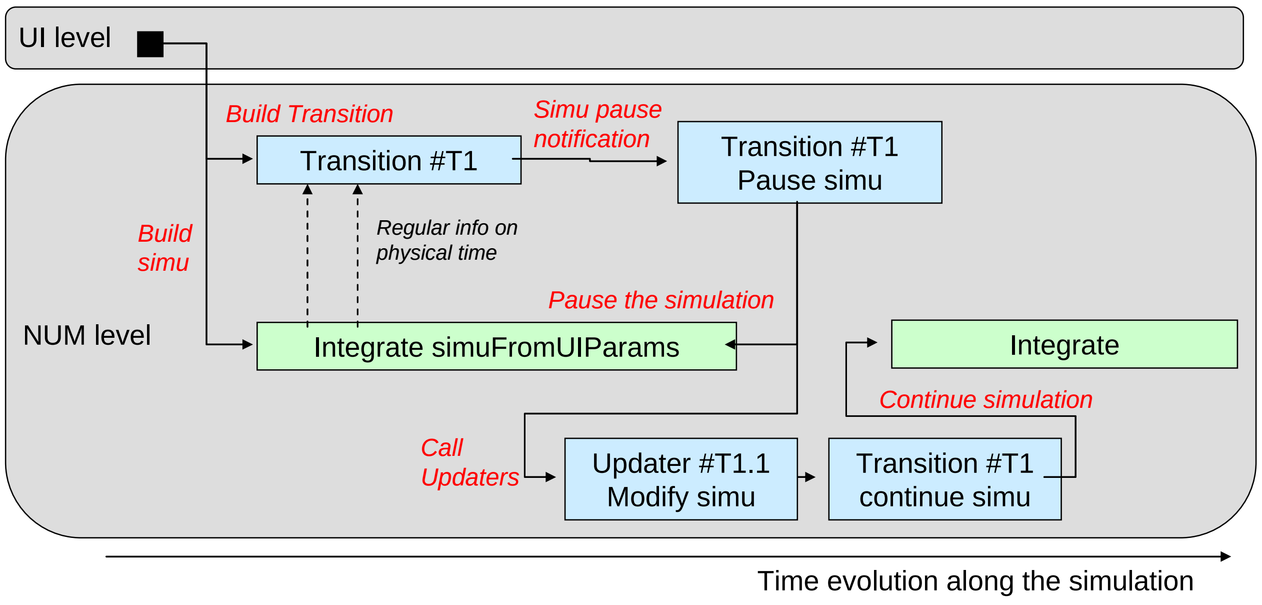

UI level
Build Transition
Simu pause notification
Transition #T1
Transition #T1 Pause simu
Build simu
Regular info on physical time
Pause the simulation
NUM level
Integrate simuFromUIParams
Integrate
Continue simulation
Call Updaters
Updater #T1.1 Modify simu
Transition #T1 continue simu
Time evolution along the simulation

Figure 39 - Logical diagram of the interaction between the simulation integration and the transitions

3.7.3. Implementation of the Transition class

The Transition interface follows the design patter Observable/Observer. The observable is the simulation (the observed parameter being the current physical time of the simulation). The observers are the realizations of the interface Transition that are added to the simulation.

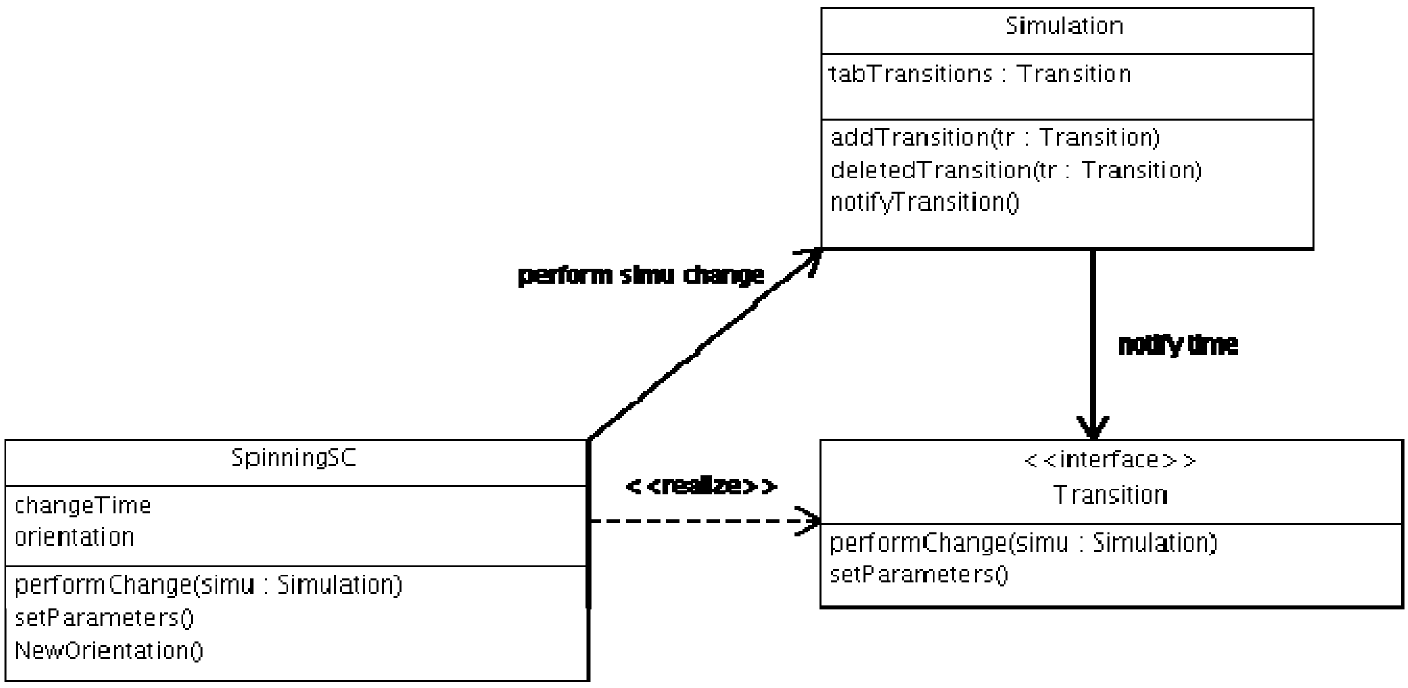
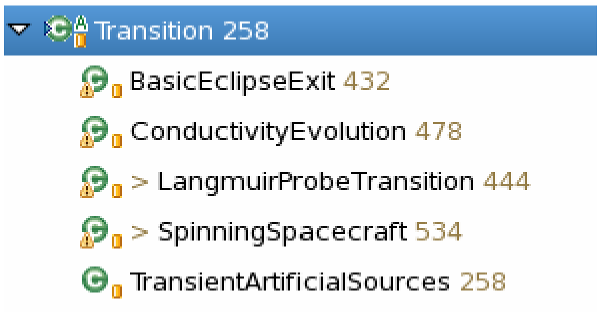


Figure 40 - UML graph of the Transition class (example for SpinningSC class)

More details of the available Transition class implementation are provided in the next table.

Table 4 - List of Transitions

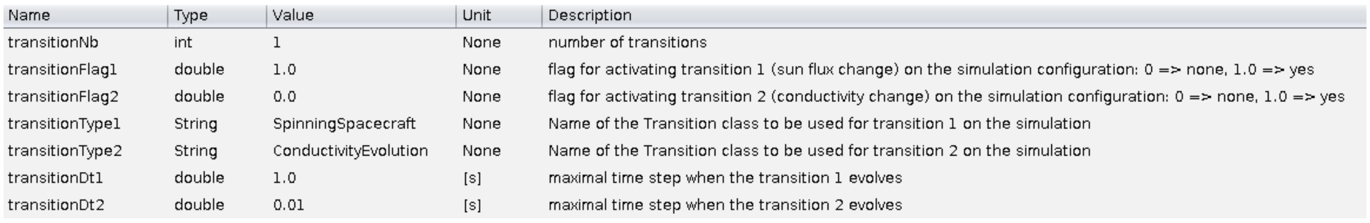


Transition 258
- BasicEclipseExit 432
- ConductivityEvolution 478
- > LangmuirProbeTransition 444
- > SpinningSpacecraft 534
- TransientArtificialSources 258

Constructors are called using global parameters given by the user:

- transitionNb defines the number of transitions
- transitionFlag# turns on/off the transition #
- transitionType# is one of the class of Table 4.
- transitionDt# controls the maximal time step allowed when the transition modifies the simulation (hence helps limiting numerical overshoots)

Table 5 - List of Global parameters associated to Transitions definition



Name	Type	Value	Unit	Description
transitionNb	int	1	None	number of transitions
transitionFlag1	double	1.0	None	flag for activating transition 1 (sun flux change) on the simulation configuration: 0 => none, 1.0 => yes
transitionFlag2	double	0.0	None	flag for activating transition 2 (conductivity change) on the simulation configuration: 0 => none, 1.0 => yes
transitionType1	String	SpinningSpacecraft	None	Name of the Transition class to be used for transition 1 on the simulation
transitionType2	String	ConductivityEvolution	None	Name of the Transition class to be used for transition 2 on the simulation
transitionDt1	double	1.0	[s]	maximal time step when the transition 1 evolves
transitionDt2	double	0.01	[s]	maximal time step when the transition 2 evolves

Detailed transient phases depend on the transition itself:

- ASCII tables located in the NumKernel/Input/ repository of the simulation are sometimes used to define the basic exit eclipse, the spinning spacecraft and the activation of artificial sources (2 header lines are used for comments).
- The material conductivity evolution, aiming at representing changes of material bulk conductivity versus time (to mimic eclipse exit for instance), is a tabulated material property.
- The Langmuir Probe transition is automatically generated by a LangmuirProbe instrument.
- Details are given in Table 6.

Importantly is the definition of times used to calculate and update the value of changing parameters. Pending on the transition itself, the update can be done:

- regularly, i.e. at each simulation time step: *BasicEclipseExit* , *ConductivityEvolution*, *TransientArtificialSources*.
- or only at pre-defined check point: *SpinningSpacecraft*, *LangmuirProbeTransition*. It is useful when the transition is costly (for instance spinning spacecraft) or when changes are really discrete (Langmuir probe).

The *transitionDt#* is always applied to *simulationDt* when arrived at a transition check point in order to control the time step even for large changes of parameter values.

The list of transitions is added to the simulation during the initialization of *SimulationFromUIParam*s. They are controlled by a *transitionObserver* which role is to determine what is the next check point (among all transitions), to control that the simulation integration actually stops at the next check point and to notify corresponding transition(s) by calling the *performTransition*(time, dt) method.

The design of all current transitions is given in the next table.

Table 6 - Transitions description and setting

Transition class	Action	Updater classes used	ASCII file name	ASCII file column 1	ASCII file column 2	Tabulated material property	Instrument
BasicEclipseExit	Performs a progressive change in sun flux magnitude (not direction)	*SunFluxIntensity Updater*	BasicEclipseExit.txt	time (Unit: [s])	relative sun flux wrt to initiall definition [-]		
ConductivityEvolution	Performs prog. change in bulk conductivity using the tabulated BUCT material property	*RCCabsSCUpdater*	N/A	N/A	N/A	BUCT [ohm^{-1}.m^{-1}] vs. Time [s]	
LangmuirProbeTransition	Perform potential sweep		N/A	N/A	N/A	N/A	LangmuirProbe
SpinningSpacecraft	Rotates progressively the ambient particle injection basis, sun flux, B and V cross B field	*SunFluxUpdater VcrossBfieldUpdater*	SpinningSpacecraft.txt	X-coord of spin axis Y-coord of spin axis Z-coord of spin axis SC Ang. vel. (rad/s) Check time period (s)	N/A	N/A	N/A
TransientArtificialSources	modifies the flux of a source	*SourceFluxUpdater*	TransientArtificialSourceX.txt where X is the Id of source	time (Unit: [s])	relative source flux wrt to initial definition [-]	N/A	N/A
TransientArtificialSources	modifies the flux of a sub-source	*SourceFluxUpdater*	TransientArtificialSourceX.Y.txt wher X is the Id of the mother source and Y the Id of the sub-source	time (Unit: [s])	relative source flux wrt to initial definition [-]	N/A	N/A

Remark for the SpinningSpacecraft transition :

The Gmsh geometry file is defined in the referential frame linked to the spacecraft. When the spacecraft rotates with an angular velocity ω, the orientation of injected particles, sun direction and magnetically induced electric field are changed. The rotation of a space vector **U** is given by :

$\vec{V} = R(\vec{U},\theta) = \vec{U}\cos\theta + (1-\cos\theta)(\vec{U}\cdot\vec{N})\vec{N} + \vec{N}\otimes\vec{U}\sin\theta$, where **N** is the rotation axis and $\theta = \omega t$

The design of all current updaters is given in the next table.

Table 7 - List of Updaters used in transitions

Updater class	Action	Inputs
SunFluxIntensityUpdater	Changes the intensity of the sun/photoEmission flux	new photoemission flux intensity as a ScalSurfField on spacecraft
SunFluxUpdater	Changes the intensity and direction of the sun /photoemission flux	sunX, sunY and sunZ components of the sun flux as defined in UI global parameters (|Sun| = 1 at 1 AU).
RCCabsSCUpdater	Modifies the RCCabsSC spacecraft circuit of the simulation	new sets of material generic parameters
VcrossBfieldUpdater	Modifies the electric field induced by the motion of spacecraft in a magnetic field, both in particle pusher and in spacecraft potential boundary condition.	VcrossB field has a space vector
SourceFluxUpdater	Modifies the flux of a volume distribution created from an artificial source on the spacecraft (the surface distribution and sampler)	Artificial volume distribution to update, Source current, Local Id of the source active on each mesh surface, Id of this source

3.7.4. Applications

Some illustrations of transitions are presented in this paragraph.

The **eclipse exit** can be simulated by combination of a *BasicEclipseExit* and *ConductivityEvolution* transitions, as shown in Figure 41.

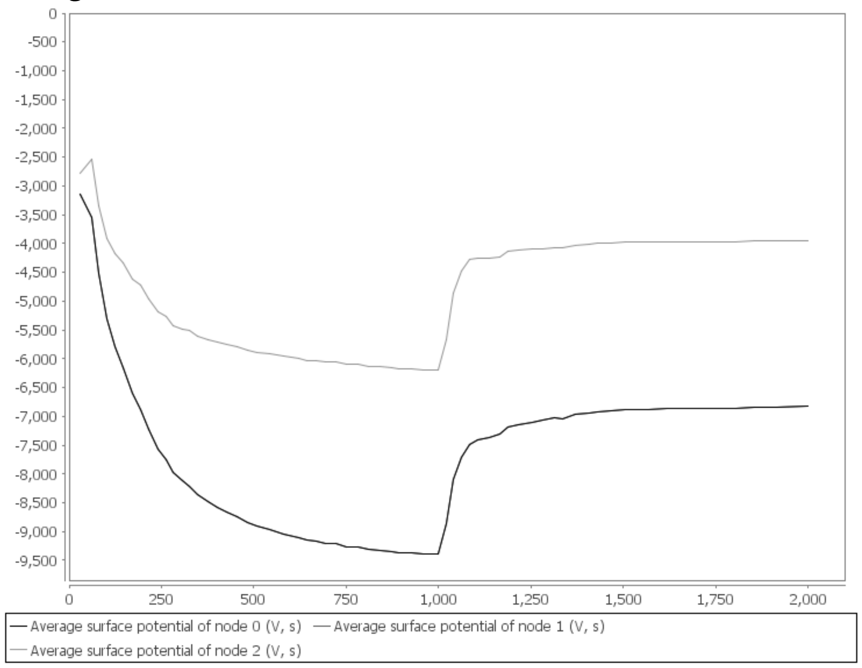


Figure 41 - Transient phase of eclipse exit. Node 2 refers to the solar panels sunlit faces of a GEO spacecraft. Exit from eclipse occurs at time t = 1000 s and lasts 60 s and consists in progressive sun flux and bulk conductivity evolution.

Figure 42 illustrates a spherical spacecraft (radius 0.1 m) covered half with ITO and half with kapton immersed in a maxwellian plasma (temperature 0.1 eV, density 1e9 m^{-3} and drift of 7000 m/s along X), including photoemission (Sun at 1 AU in Z-direction), with a spin of 2π rad/s around the Y-axis.

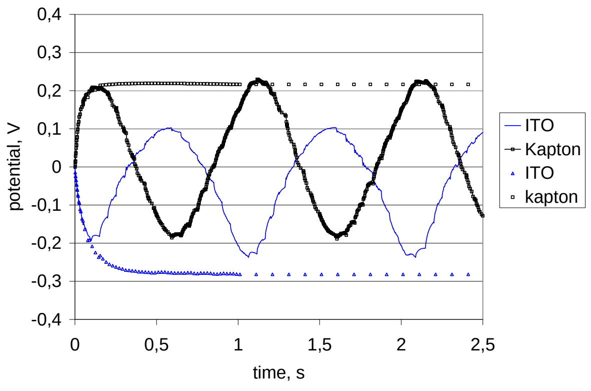
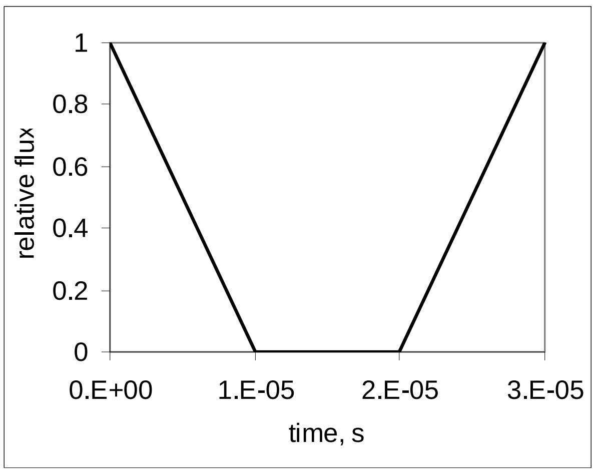


Figure 42 - Left: Potentials on a spinning spacecraft (lines) compared to a fixed attitude spacecraft (dots). The spin velocity is one rotation per second. Right: probe potential when

The last example of Figure 43 concerns the activation/deactivation of a particle source on spacecraft.



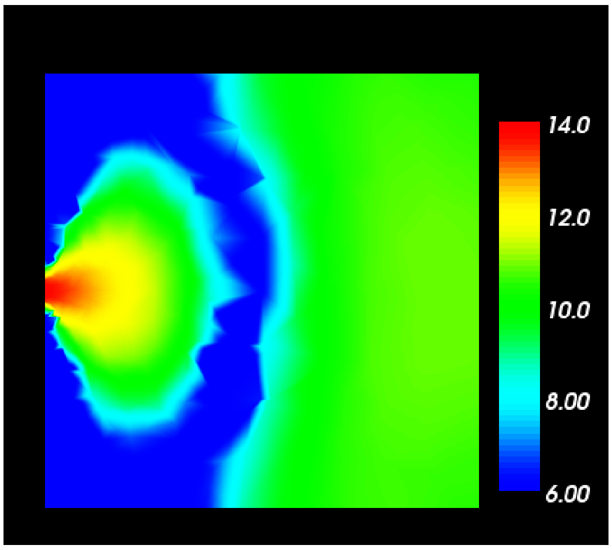


Figure 43 - Progressive artificial source de-activation and re-activation. The spacecraft surface is on the left of the simulation box.

3.8. Magnetic field

The magnetic field capabilities of previous SPIS versions were:

- Uniform and constant
- Runge-Kutta Cash-Karpe (RKCK) method to solve the particle trajectories

New items are:

- Iterative pusher for particle trajectory computing
- Magnetically induced electric field due to spacecraft motion
- Finer control of the processes

3.8.1. Iterative pusher

The fifth order RKCK method being precise but quite expensive in term of CPU time, an upgrade of the particle pusher in the case of a magnetic field has been performed. In each cell (without any thin element), the electric field is constant. As the magnetic field is also constant, the particle trajectory is analytical. However, the interception the trajectory with a surface is not analytical. Hence the calculation of the particle trajectory with the cell surface boundaries is calculated through a dichotomy method.

Rationale:

Particle motion writes: $\frac{d\vec{V}}{dt} = \frac{q}{m}\left(\vec{E} + \vec{V} \otimes \vec{B}\right)$ which is decomposed in a component parallel to the magnetic field (//) and in a component perpendicular to the magnetic field (⊥):

$$\frac{d\vec{V}_{//}}{dt} = \frac{q}{m}\vec{E}_{//}$$

$$\frac{d\vec{V}_{\perp}}{dt} = \frac{q}{m}\left(\vec{E}_{\perp} + \vec{V}_{\perp} \otimes \vec{B}\right)$$

The local orthonormal basis (X, Y, Z) is defined such as

- $\vec{E}_{\perp} = E_{\perp}\vec{X}$
- At t = 0 s, $\vec{V}_{\perp}(0) = \vec{V}_{\perp,X}(0) + \vec{V}_{\perp,Y}(0) = V_{\perp,X}(0)\vec{X} + V_{\perp,Y}(0)\vec{Y}$

At any time t, one can demonstrate that the velocity $\vec{V} = V_X\vec{X} + V_Y\vec{Y} + V_Z\vec{Z}$ writes:

$$V_X(t) = V_X(0)\cos(\omega t) + \frac{q}{|q|}\left(V_Y(0) + \frac{E_{\perp}}{B}\right)\sin(\omega t)$$

$$V_Y(t) = \left(V_Y(0) + \frac{E_{\perp}}{B}\right)\cos(\omega t) - \frac{q}{|q|}V_X(0)\sin(\omega t) - \frac{E_{\perp}}{B}$$

$$V_Z(t) = \frac{q}{m}E_{//}t + V_Z(0)$$

and the helicoidal position:

$$X(t) = \frac{V_X(0)}{\omega}\sin(\omega t) + \frac{q}{|q|\omega}\left(V_Y(0) + \frac{E_{\perp}}{B}\right)(1 - \cos(\omega t)) + X(0)$$

$$Y(t) = \frac{1}{\omega}\left(V_Y(0) + \frac{E_{\perp}}{B}\right)\sin(\omega t) + \frac{q}{|q|}\frac{V_X(0)}{\omega}(\cos(\omega t) - 1) - \frac{E_{\perp}}{B}t + Y(0)$$

$$Z(t) = \frac{1}{2}\frac{q}{m}E_{//}t^2 + V_Z(0)t + Z(0)$$

where $\omega = \frac{|q|B}{m}$

Algorithm:

A new method *crossTetraUniformEwithDichotomy* of *ComplexPusher* class has been implemented. It takes the same arguments as inputs as previous methods used to calculate particle trajectory and interception by tetrahedron surfaces.

Remark: this method also applies to the case of interception by a thin wire by dichotomy. This is we describe here the algorithm also for wires.

First, the local basis $(X, Y, Z)_{local}$ is constructed from the local value of B and E. The matrix *M* permitting to pass from SPIS natural basis $(X, Y, Z)_{SPIS}$ to local basis is constructed. M: $(X, Y, Z)_{SPIS} \rightarrow (X, Y, Z)_{local}$.
In order to avoid particle exiting and re-entering the cell in tha same time step, the time step is initially limited to characteristics times:

- CFL-like (characteristic time to cross a fraction of cell with the initial velocity)
- Magnetic field condition (fraction of gyro-frequency)
- Acceleration by $E_{//}$.

For each sub time step:

- Calculation of the new position in local and SPIS natural coordinates
- Test particle position wrt to thin wires in the current cell. 3 possibilities :
    - particle is outside any wire
    - particle is on a wire surface : the trajectory is stopped
    - particle is inside a wire: we go backward in time and the current time step is divided by 2
- Test particle position wrt to cell surfaces so as to determine if the particle is inside/outside the cell
    - particle is far outside a cell : go back to previous time and divide the time step by 2
    - particle is inside a cell : go to the new position and keep the same time step, except if the initial time step has been divided once (then divide it by 2)
    - particle is outside the cell, with only one negative barycentric coordinate, this coordinate being small compared to pusher tolerance : the exit surface is identified
- The position on thin wires or surfaces is controlled by a relative tolerance level *iterativePusherRelTolPos* (default value is 1e-3), i.e. the particle is collected if its barycentric coordinates are smaller than the tolerance level.

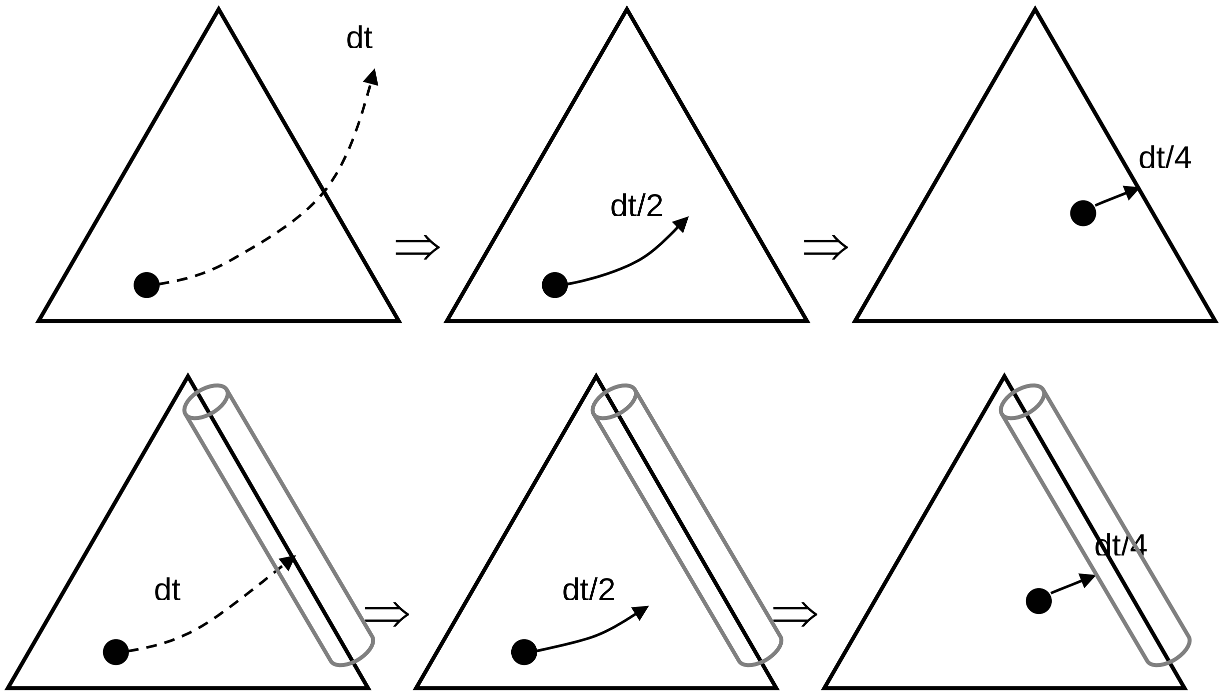


Figure 44 - Illustration of the dichotomy algorithm used for particle pushing in case of magnetic field. Top: general design; Bottom: application to the presence of an unmeshed thin wire.

Illustration:

Results are compared to the non-regression case of the SPIS-Maintenance activity *SPIS_NRC_EcrossB.* Protons of 10 eV drift along in X-axis (source located at Y = 0.2 m with a current density of 1e-7 A/m^2 (divided by 100 wrt NRC). Boundary conditions in +X and -X planes impose a macroscopic electric field of 10 V/m in X-axis. A Magnetic field of -4e-4 T is imposed along Z-axis.
Results agree with the theoretical intersection of the jet with the plane X = 1 m at position Y = 0.5866 m. The performance for a simulation time step of 0.6 µs, a duration of 0.2 ms, 1 million of super particles and 17897 tetrahedron cells is:

- RKCK: 19 min CPU
- Dichotomy: 12 min CPU

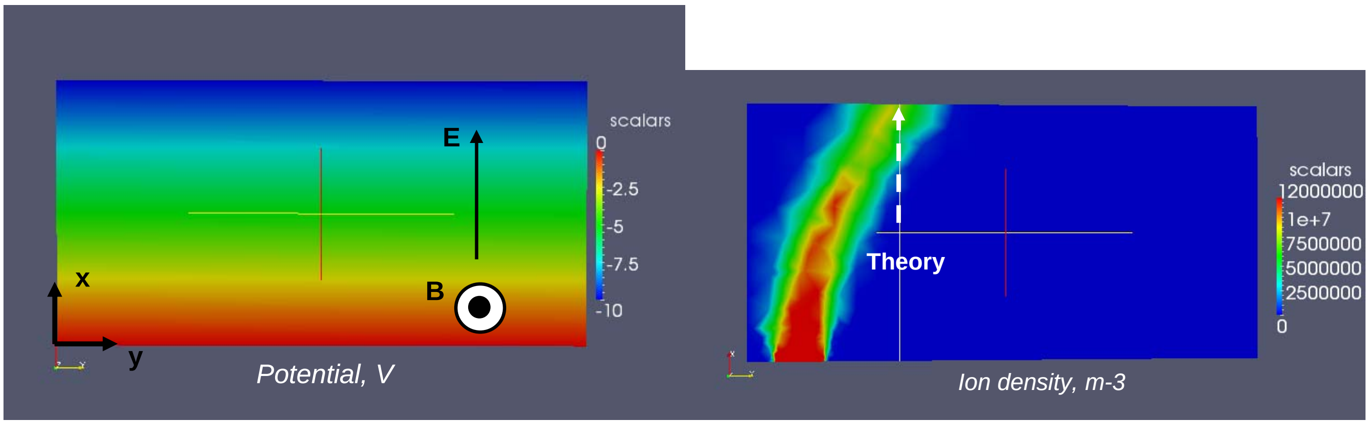


Table 8 - Simulation of an ion jet dynamics immersed in E cross B field. Comparison with theory.

Input parameters:
The new global parameters permit a better control of the particle pusher.

Table 9 - New global parameters for particle pusher settings

KeyName	Value Type	Default Value	Unit	Description	Verbosity Level	Comment
magnetizedPlasmaFlag	int	1	[-]	flag for taking account the effect of the magnetic field and of the magnetically induced electric field (due to spacecraft motion) on particle trajectories. 1: yes; 0: no (un-magnetized plasma)	MEDIUM	In case magnetic field is turned ion by user, particles will be considered as magnetized
BFieldIterativePusher	int	1	None	flag for particle pusher method used in case of magnetic field (0: RKCK algorithm from spis 4.3; 1: Dichotomy method from spis 5)	ADVANCED	In case of B field
iterativePusherAbsTolPos	double	0.00001	[m]	precision of iterative particle pusher (RKCK method): absolute tolerance position	EXPERT	
iterativePusherAbsTolVelo	double	1E+12	[m/s]	precision of iterative particle pusher (RKCK method): absolute tolerance velocity	EXPERT	
iterativePusherRelTolPos	double	0.001	None	precision of iterative particle pusher (either RKCK or dichotomy method): relative tolerance position	EXPERT	
iterativePusherRelTolVelo	double	0.001	None	precision of iterative particle pusher (RKCK method): relative tolerance velocity	EXPERT	

3.8.2. *Electric field induced by spacecraft motion in electric field*

Rationale:

The change of space referential from the plasma referential to the spacecraft referential leads is accompanied by a change in the electromagnetic field. Let note (ϕ, **E**, **B**) and (ϕ', **E'**, **B'**) the electric potential, the electric field and the magnetic field in the referential R of the plasma and in the referential R' of the spacecraft, respectively. The spacecraft velocity in R is noted $\mathbf{V_{SC}}$. In the non-relativistic limit:

$$\vec{B}' = \vec{B}$$

$$\vec{E}' = \vec{E} + \vec{V}_{SC} \otimes \vec{B}$$

$$\vec{E} = -\nabla\phi \quad , \quad \vec{E}' = -\nabla\phi'$$

In the referential R of the plasma

- The undisturbed electric field, i.e. in the absence of the spacecraft and neglecting the electric field generated by an external source (as Earth for example), is null
- Due to the Hall effect, a potential gradient establishes in a conductive drifting spacecraft and the electric field inside the conductor becomes $-\vec{V}_{SC} \otimes \vec{B}$

In the referential *R'* of the spacecraft in translation of $\mathbf{V_{SC}}$ with respect to *R*:

- The electric field in an undisturbed plasma, becomes $\vec{E}' = \vec{E}_\infty = \vec{V}_{SC} \otimes \vec{B}$
- Conductive elements are isopotential

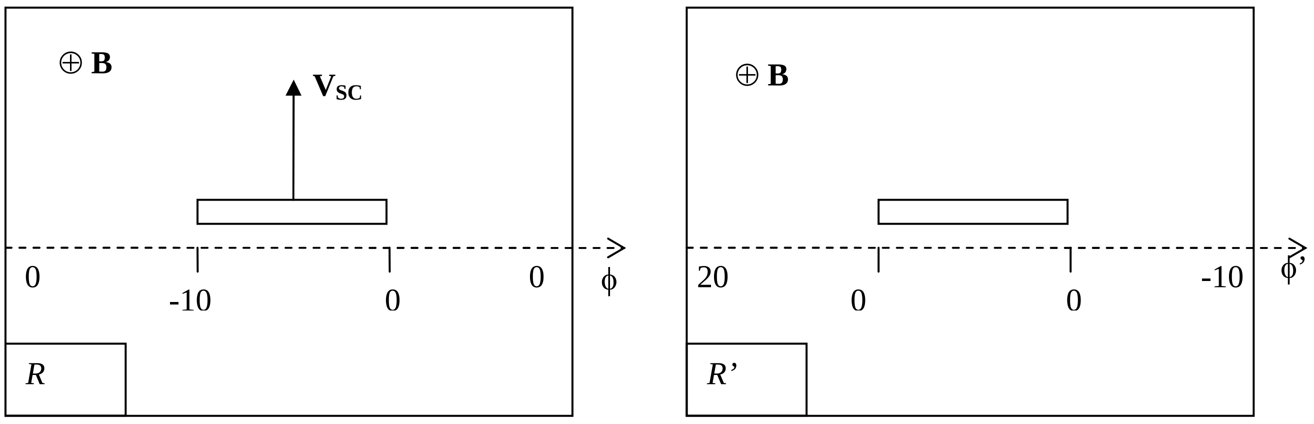


Figure 45 - Illustration of the potential generated by a spacecraft drifting in a magnetized plasma; in the referential of the plasma (left) and in the referential of the spacecraft (right)

In the example of Figure 45, the electric field generated by the (1m long) spacecraft drift is -10 V/m in the referential *R*. In the referential *R'* of the spacecraft, an additional electric field of -10 V/m is added to the undisturbed electric field **E** (the plasma domain is 30 m long). This V cross B field can clearly be taken into account either placing the problem either in the referential *R*, either in the referential *R'* or, as a third solution: **a mix of both**.

Implementation:

Constraints on the reference frame choice are:

- The spacecraft mesh lives on the referential *R'* of the spacecraft.
- The boundary conditions are simple to define in *R* and more complicated in *R'*.
- The spacecraft circuit is well-defined in *R'*.
- The ambient particle velocities **v** are naturally defined in *R*. In *R'*: **v'** = **v** - $\mathbf{V}_{\mathbf{SC}}$.
- The particles emitted by the spacecraft are naturally defined in *R'*.

Both data living on *R* and *R'* are used to benefit from the past implementations of SPIS and avoid a too high level of complexity of the developments:

- A new variable *scVelocity* (of type DimVect) is added to the class *SC* defining the spacecraft
- Poisson equation is solved in the referential *R* of the plasma: **E** and ϕ are used
  - The external boundary conditions are unchanged with respect to past SPIS implementations
  - The boundary conditions on the spacecraft are changed to fit the Hall effect: the Dirichlet potential ϕ is calculated on each point of coordinates **x** of the spacecraft surface using the potentials ϕ' in the referential of the spacecraft R': $\phi = \phi' + \left(\vec{V}_{SC} \otimes \vec{B}\right).\vec{x}$
  - ϕ is defined to an arbitrary constant depending on the reference position $\mathbf{x_0}$ leading to ϕ = ϕ'. In order to have as much as possible negative potentials on the spacecraft, it is decided to set $\mathbf{x_0}$ such as max(ϕ) = 0 (this choice has no impact on the final solution since the current adapts to impose the same final equilibrium potential).
  - The Boltzmann distribution equation (used for density and in the Poisson non-linear solver) is unchanged since such species are in equilibrium with ϕ.
  - In the case of a spacecraft rotating at angular velocity ω around the spin axis $\mathbf{S}_{\mathbf{SPIN}}$, the second term of the equation is solved using the same approach as described in paragraph 83

$$\left(\vec{V}_{SC} \otimes \vec{B}\right)(t) = \vec{E}_{VB}(t)$$

$$\theta = \omega(t - t_0)$$

$$\vec{E}_{VB}(t) = -\left[\cos\theta \cdot \vec{E}_{VB}(t_0) + (1-\cos\theta)\left(\vec{E}_{VB}(t_0)\cdot\vec{S}_{SPIN}\right)\vec{S}_{SPIN} + \sin\theta\left(\vec{S}_{SPIN} \otimes \vec{E}_{VB}(t_0)\right)\right]$$

- The particle dynamics is solved in the referential *R'* of the spacecraft since their positions **x'**, velocity **v'** as well as their collection, emission currents on spacecraft, are all defined in the frame of the spacecraft mesh. However, we use E of *R* to calculate the transport:
  - The particle pusher is modified taking into account the equation of the dynamics in *R'*:

$$\frac{d\vec{x}'}{dt} = \vec{V}'$$

$$\frac{d\vec{V}'}{dt} = \frac{q}{m}\left(\vec{E} + \vec{V}' \otimes \vec{B} + \vec{V}_{SC} \otimes B\right)$$

  - The injection of particles is changed with respect to the previous injection defined in *R*. As the velocity distribution is initialized by users in the referential *R* at rest, a drift velocity related to the spacecraft motion wrt to *R* is added.
  - In case of spinning spacecraft, the same correction is applied.

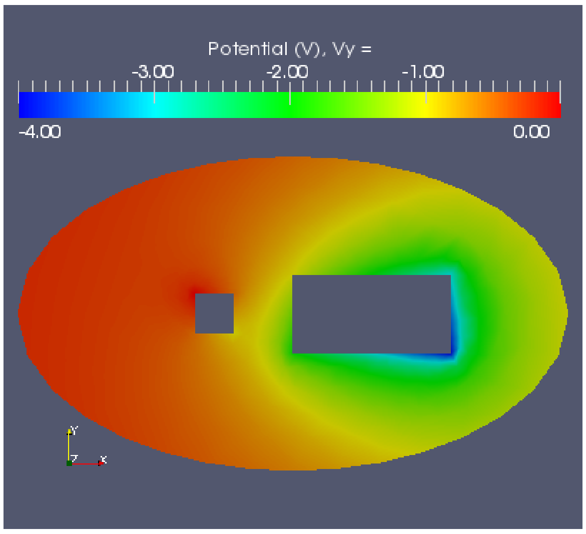
- The spacecraft circuit is unchanged, i.e. solved in the reference frame *R'* of the spacecraft. It calculates the surface potentials ϕ' needed by the Poisson solver as a spacecraft boundary condition.

Input parameters:

Table 10 - New global parameters for VcrossB field

KeyName	Value Type	Default Value	Unit	Description	Verbosity Level	Comment
scVeloCrossBFlag	int	1	[-]	flag to take account the effect induced by the spacecraft drift on the spacecraft surface potential (in the reference frame of the plasma)	ADVANCED	If magnetic field and drifting spacecraft, then the electric field induced by the motion of the spacecraft is turned on.
scVeloX	double	0	[m/s]	x-component of the spacecraft velocity in the reference frame of the plasma at rest	ADVANCED	
scVeloY	double	0	[m/s]	y-component of the spacecraft velocity in the reference frame of the plasma at rest	ADVANCED	
scVeloZ	double	0	[m/s]	z-component of the spacecraft velocity in the reference frame of the plasma at rest	ADVANCED	

Illustrations:



Electric potential around a spacecraft flowing at velocity 7500 m/s in a magnetic field of 5e-5 T. Potential difference on this 10 m long spacecraft is about 4 V. The plasma is not considered here

The electric field induced by the motion of a spacecraft flowing in LEO condition is presented in the figure on the left. For this 10 m long spacecraft, a potential drop of about 4 V is obtained on spacecraft surface. On the International Space Station the potential difference can attain 50 to 100 V typically. The effect on populations is not simulated here but the electric field is strongly affected.

The second illustration concerns a spherical probe, with comparison to results obtained on another code (R. Marchand, "PTetra, a Tool to Simulate Low Orbit Satellite–Plasma Interaction", IEEE Trans. Plasma Sci., Vol 40, N°2, pp. 217-229, 2010). In this example, electrons and ions are submitted to a magnetic field of 3×10^{-4} T in the z-direction. Low temperature electrons are magnetized and ions are practically unmagnetized (large gyroradius). Debye length is equal to probe radius, i.e. 2 cm. The probe velocity of -7500 m/s in the y-direction induces a potential drop of 0.19 V on probe surface, in the plasma referential at rest.
The probe floating potential is -0.4 V. The current density on the probe is similar to the results obtained previously. The asymmetric 3D structure is well reproduced, see Figure 46.

Quantity	Value
Temperature	0.2 eV
Electron/ion density	$2{,}8\times10^{10}$ #/m^3
Debye length	0.02 m
Sphere radius R	0.02 m
Number of tetrahedrons	~79,433
Simulation box diameter	0.20 m
Number of macro-particles	~500,000
Bz	3×10^{-4} T
Electron Gyro Radius / R	0.18
Ion Gyro Radius / R	7.6
Spacecraft velocity Vy	- 7500 m/s

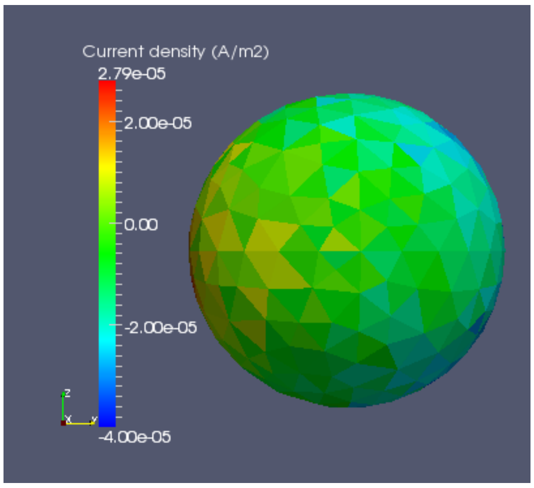


Figure 46 - Current density on a probe flowing in a magnetized plasma

3.9. Performance and accuracy

Performance and accuracy has been improved by a multi-threading approach, the adaptation of boundary conditions in term of electric field and matter injection, and by providing users with a large set of advanced monitoring capabilities.

3.9.1. Varying number of super particles

Due to meshing constraints (unstructured mesh with local refinement close to small spacecraft surfaces), super particle statistics can vary significantly in the simulation boxes. As particle are injected at the frontier of the box, their number per cell decreases when getting closer to the spacecraft. The user can choose to increase the number of super particles with the global parameter
The code detects first areas that need to be repopulated with super-particles. The information is tracked toward the particle sampler on surface boundaries, by using the initial positions of particles that succeeded in reaching the target areas. The sampler multiplies the number of super particles.

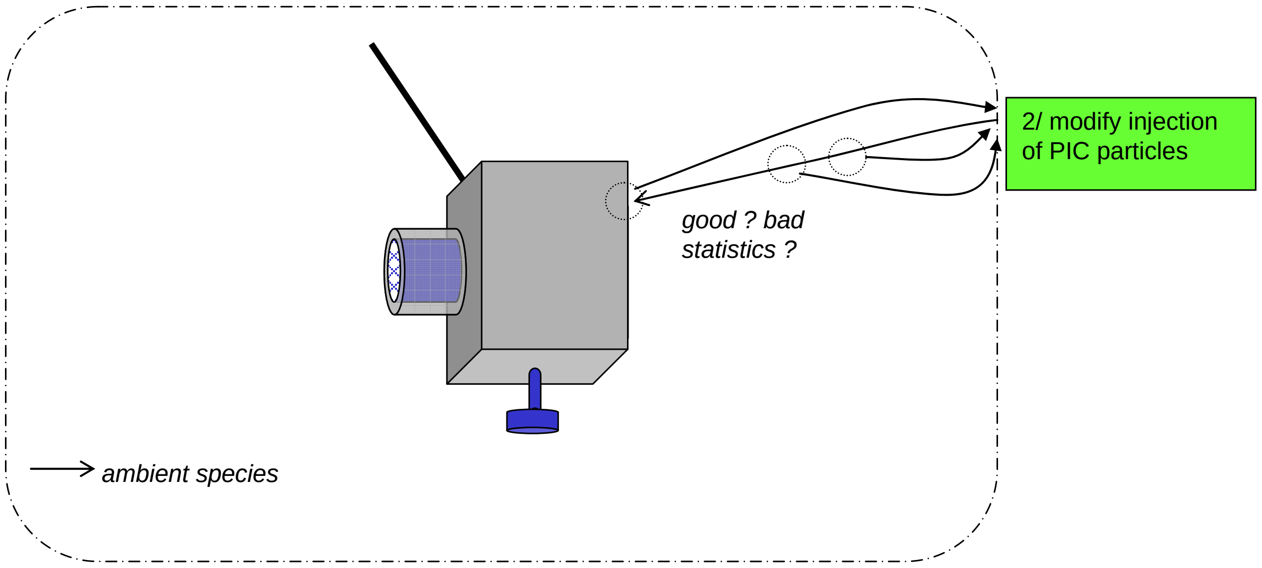


Figure 47 - Principle of varying number of super-particle injection

Algorithm:

- Keep the information of each particle origin using an upgrade of the *RichPartList* class also used for particle detection, i.e. the **tracking mode**. in which particle initial properties are saved (position, velocity).
- Indicate a target for statistics as number of SP per volume node (some nodes with a higher target objective, as e.g. close to spacecraft, thin wire)
- At each integration step, collect the statistics of SP per volume node as a matrix M(injection surface, volume node) relating the surfaces of injection to the volume nodes.

- Compare the current super-particle number to the target objective on each cell and generate a local densification of the SP number to be injected locally on each boundary surface for the next time step
- Keep 100 % of the old injection and add X % of optimized particles, X being controlled by user with the global parameter *pop#Optimization* (example: 0.5 => add 50 % of optimized particles to the original liste). This feature is only applicable to **Generic PIC population**, see section 3.4.
- Inject the modified number of SP

3.9.2. External boundary conditions upgrade

The precision of the plasma solver has been improved upgrading the boundary conditions of the external frontier.

3.9.2.1. Matter boundary conditions

At external boundaries, the potential is regularly different from zero volts, meaning that frontiers can be inside the plasma sheath around the spacecraft. As a result, the injection of a constant distribution assuming an undisturbed plasma is only approximate. This can explain the errors of some 3 to 5% often observed when comparing previous SPIS version with well-known characteristics of spherical or cylindrical probes.

In the new version, fluxes and distribution functions at boundaries are modified during injection, taking into account the local potential.

Algorithm in 3D:
The flux of a Maxwellian repulsed specie becomes:

$$F_{repulsed} = n_0\sqrt{\frac{eT}{2\pi m}} \times \exp(\chi)$$
$$\text{with } \chi = -\frac{q}{e}\frac{V}{T}$$

Assuming Liouville theorem assumptions (isotropic specie, all trajectories in phase space filled), the flux of a Maxwellian attracted specie becomes:

$$F_{OML,3D} = n_0\sqrt{\frac{eT}{2\pi m}} \times [1+\chi]$$

$$f(\mathrm{v}).d\mathrm{v}_T = \begin{cases} \dfrac{m\mathrm{v}^2}{2eT} > \chi, n_0 \times 4\pi\exp(\chi)\times\left(\dfrac{m}{2\pi eT}\right)^{3/2}\mathrm{v}^2 \\ \qquad\qquad \times\exp\left(-\dfrac{m(\mathrm{v}^2)}{2eT}\right)d\mathrm{v} \\ \dfrac{m\mathrm{v}^2}{2eT} < \chi, 0 \end{cases}$$

Algorithm in 2D:

For the attracted specie:

$$F_{OML,2D} = \frac{2}{\sqrt{\pi}} \times n_0 \sqrt{\frac{eT}{2\pi m}} \times \left[\sqrt{\chi} + \frac{\sqrt{\pi}}{2} \exp(\chi) \operatorname{erfc}\left(\sqrt{\chi}\right) \right]$$

$$f(\mathrm{v}_r, \theta, \mathrm{v}_z).d\mathrm{v}_T = \begin{cases} \dfrac{m\mathrm{v}_r^{\ 2}}{2eT} > \chi, n_0 \exp(\chi) \times \left(\dfrac{m}{2\pi eT} \right)^{3/2} \mathrm{v}_r \\ \qquad \times \exp\left(-\dfrac{m\left(\mathrm{v}_r^{\ 2} + \mathrm{v}_z^{\ 2}\right)}{2eT} \right) d\mathrm{v}_r \, d\theta \, d\mathrm{v}_z \\ \dfrac{m\mathrm{v}_r^{\ 2}}{2eT} < \chi, 0 \end{cases}$$

Illustration:

An example of the precision obtained with this improvement is shown later ion in section

3.9.2.2. Electric field boundary condition

The algorithm consists in automatically and locally shifting between two regimes

- o If the external boundary is outside the sheath, the presheath model is used $\phi \sim 1/r^2$
- o Else: vacuum-like condition $\phi \sim 1/r$

The updater *SheathOrPresheathPoissonBCUpdater* is attached to the simulation. Its role is to update dynamically the Poisson boundary conditions following the plasma parameters. If the local potential $|\phi|$ is smaller than the electron temperature kT_e/e then the pre-sheath model is used, else the vacuum-like condition is used.

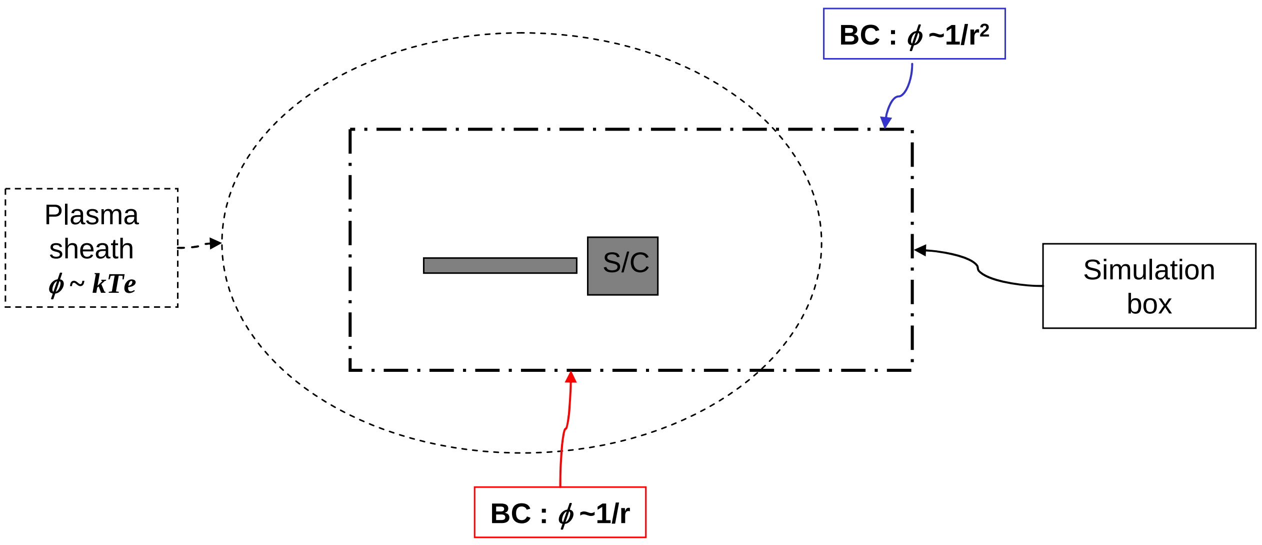


Figure 48 - Overview of boundary condition automatic selection pending on location within the plasma sheath around the object

The global parameter *poissonBCParameter2* with a value of -1 activate this regime (1 =vacuum-like; 2 = pre-sheath; 0 = mixed Dirichlet Neuman condition)

3.9.2.3. Illustrative example

Characteristic of a spherical probe in a Maxwellian plasma:
Characteristics of a sphere in a maxwellian unmagnetized collisionless plasma of hydrogen has been compared to similar results (J. G. Laframboise, “Theory of spherical and cylindrical Langmuir probes in a collisionless, Maxwellian plasma at rest,” Ph.D. dissertation, University of Toronto, 1966). Two cases are considered, corresponding to Debye length: (a) two times larger and (b) equal to the probe radius. Simulation parameters are presented in next table.
Next figure shows the comparison with results from Laframboise. The agreement between the two is within 1% in all these long Debye length cases (a) and (b)

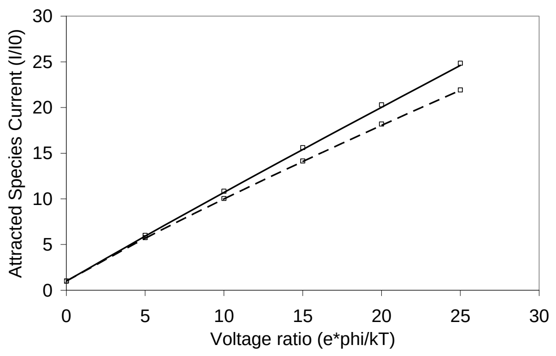
Characteristics of a cylindrical probe immersed in a Maxwellian plasma has been calculated using symmetry conditions for electric field and fluxes in the z-direction perpendicular to the cylinder axis. The attracted specie flux was calculated following the 2D OML injection algorithm instead of the default 3D OML. The agreement with Laframboise is within 1% error.

Spherical probe

Quantity	Value (a)	Value (b)
Temperature	0.5 eV	0.2 eV
Electron/ion density	6.91×10^{8} m^{-3}	2.763×10^{10} m^{-3}
Debye length	0.2 m	0.02 m
Potential	[0 to 12.5 V]	[0 to 5 V]
Sphere radius	0.1 m	0.02 m
Particle model	full-PIC	full-PIC
Number of tetrahedrons	127,759	56,726
Simulation box diameter	1.3 m	0.2 m
Number of macro-particles	430,000 to 1,500,000	193,000 to 650,000

Cylindrical probe

Quantity	Value
Temperature	0.2 eV
Electron/ion density	2.76×10^{10} m^{-3}
Debye length	0.02 m
Potential	[0-5 V]
Cylinder radius	0.02 m
Cylinder length	0.08 m
Number of tetrahedrons	45,000
Simulation box diameter	0.4 m
Number of macro-particles	460,000



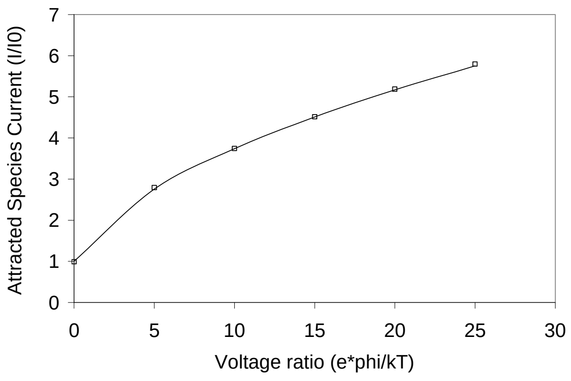


Figure 49 - Current collection on spherical (left) and cylindrical (right) probes. SPIS results are indicated with dots. Laframboise results are shown in solid line. The normalization currents are $I_0 = 4\pi n_e e R^2 \sqrt{kT/2\pi m_e}$ *and* $I_0 = 2\pi n_e e R L \sqrt{kT/2\pi m_e}$ *respectively for spherical and cylindrical probes.*

3.9.3. Multithreading

We present in this section a major improvement of SPIS efficiency in terms of CPU time cost. This enhancement concerns the multi-threading of the particle pusher. The particle pushing is one of the most time consuming tasks of the SPIS system because hundreds of thousands and possibly millions of particles are injected in the volume at each time step.

Algorithm:
The particle move method is performed in the *VolDistrib* class. In the case of a PIC volume distribution, the move method calls *ComplexPusher* objects. Both *PICVolDistrib* and *ComplexPusher* classes have been adapted to a multi-thread approach. An *ExecutorService* is associated to PIC volume distribution with a number of parallel thread controlled at UI level by the global parameter *pusherThreadNb*.

The Java language is very well adapted to the multi-thread approach. A new class *TaskOfParticlePusher* extends the *Callable* class. The data used by this class are passed to the constructor: a *ComplexPusher* and the data permitting this pusher to perform its push() method (list of particle, time step, …). The *call*() method of the *Callable* class is overridden : launch of the *ComplexPusher push*() method and charge deposit.

The *push*() method of the *ComplexPusher* has itself been modified in order to move only a fraction of the list of particles (one particle every N particle).

This method is schematically depicted in Figure 50.

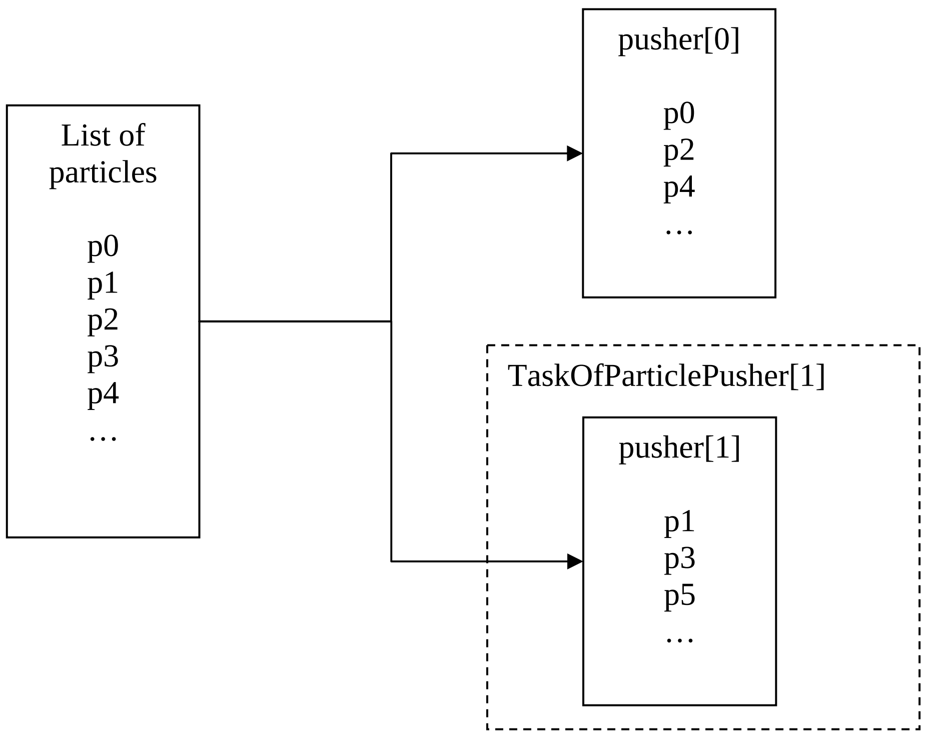


Figure 50 - Example of multi-thread approach with 2 threads. The first thread is performed in the current SPIS thread, while the second involves a new TaskOfParticlePusher

At the end of the pusher multi-thread execution, the charge deposited by each list of particles is summed up in order to calculate the electric field at next step.

Illustration:
Non-regression of the new development is shown in Figure 51, in which the results obtained with the previous SPIS version SVN:58 (based on SPIS.4.3.1) is compared to a simulation performed with the development version SVN:92. Respectively one, four and eight threads are used. The very slight differences come from the Monte-Carlo method used for particle injection during the time step.

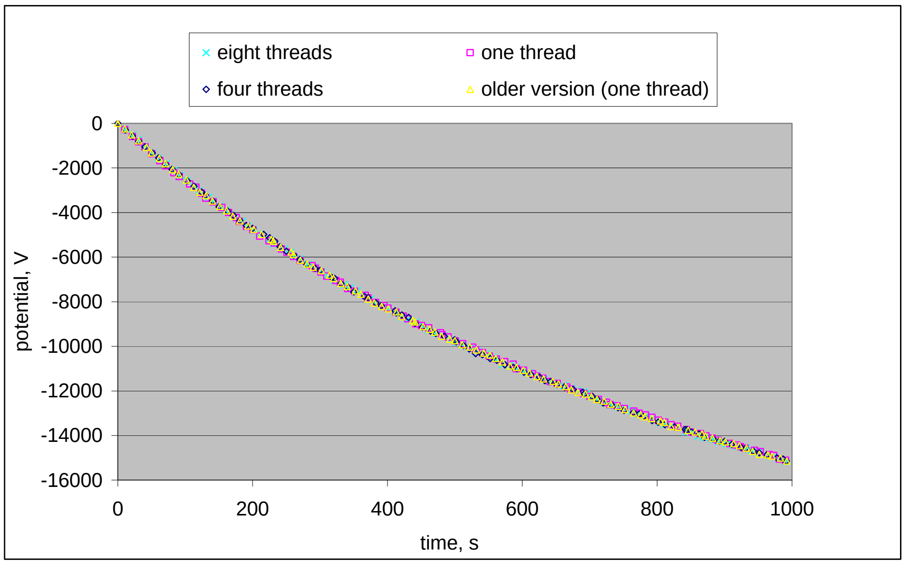


Figure 51 - Non-regression test of multi-threading method

A gain of 2.7 to 3.4 is obtained on the particle pushing with 4 threads in parallel instead of only one.
A gain of 3.2 to 4.9 is obtained on the particle pusher with 8 threads in parallel instead of only one.

The total gain depends however on the type of simulation. If the particle transport is very costly with respect to other processes, then the gain on the overall computation time can be important.

3.9.4. Circuit solver update

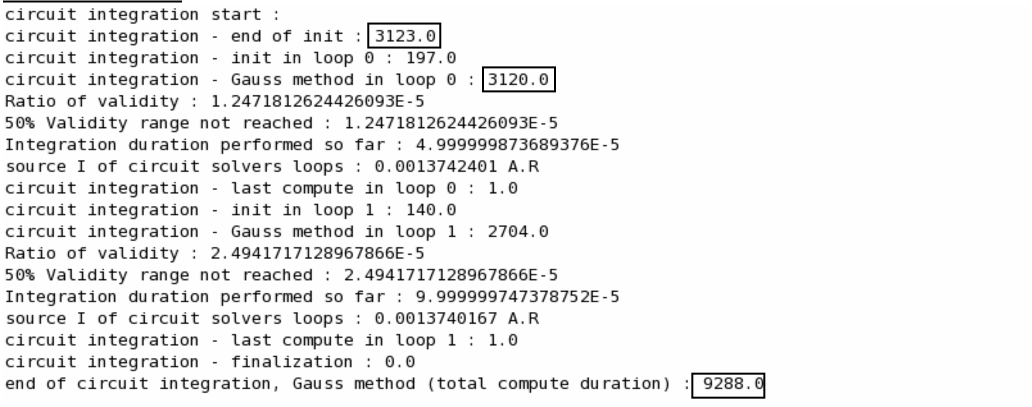
This improvement was designed in the frame of the SPIS-GEO. The spacecraft equivalent electrical circuit solver is composed of initialization/updating routines and of a linear system A.X = B where X is the solution (the potential on surface potential), A the circuit matrix and B the currents. In SPIS 4.3.1 version, a linear system solver based on a Gauss method was used. It was found to be costly when using a large number of dielectric elementary surfaces.

A linear system solver based on an iterative Conjugate Gradient Squared method (CGS) has been implemented. It has the advantage to converge in a very limited number of iterations. This new solver is controlled by the global parameter *implicitCircuitSolver* (integer, default value = 1 --> Gauss; if 0 --> use of linear solver instead).

We compared the CPU time used to solve the linear system of a 1671 node circuit: 155 ms with CGS algorithm instead of 3100 ms with Gauss method, i.e. a **gain up to 20** (sometimes 40!).

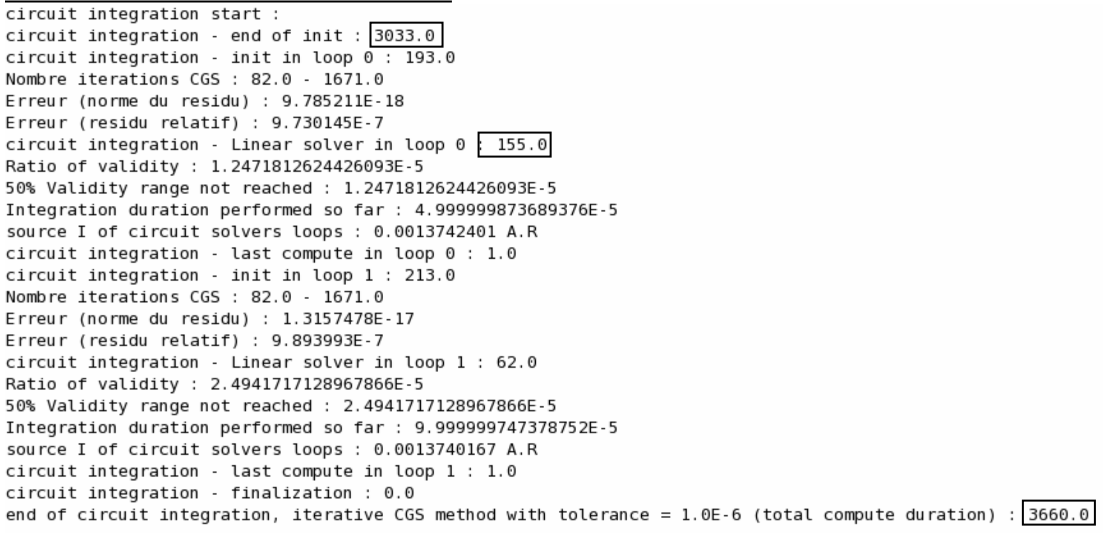
The main cost of the circuit solver is now due to the initialization of the circuit dIdV matrix at each step (3000-3100 ms). **The global gain on the circuit solver is about 2.6.**

Gauss Solver:



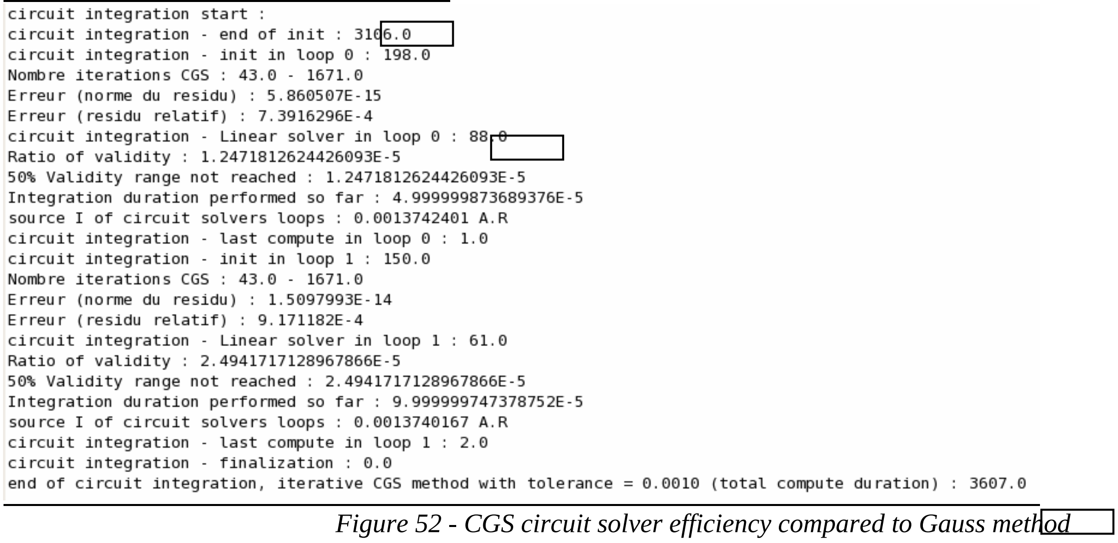
```
circuit integration start :
circuit integration - end of init : 3123.0
circuit integration - init in loop 0 : 197.0
circuit integration - Gauss method in loop 0 : 3120.0
Ratio of validity : 1.2471812624426093E-5
50% Validity range not reached : 1.2471812624426093E-5
Integration duration performed so far : 4.999999873689376E-5
source I of circuit solvers loops : 0.0013742401 A.R
circuit integration - last compute in loop 0 : 1.0
circuit integration - init in loop 1 : 140.0
circuit integration - Gauss method in loop 1 : 2704.0
Ratio of validity : 2.4941717128967866E-5
50% Validity range not reached : 2.4941717128967866E-5
Integration duration performed so far : 9.999999747378752E-5
source I of circuit solvers loops : 0.0013740167 A.R
circuit integration - last compute in loop 1 : 1.0
circuit integration - finalization : 0.0
end of circuit integration, Gauss method (total compute duration) : 9288.0
```

LinSolveDoublePrecision tol 1e-6:



```
circuit integration start :
circuit integration - end of init : 3033.0
circuit integration - init in loop 0 : 193.0
Nombre iterations CGS : 82.0 - 1671.0
Erreur (norme du residu) : 9.785211E-18
Erreur (residu relatif) : 9.730145E-7
circuit integration - Linear solver in loop 0 : 155.0
Ratio of validity : 1.2471812624426093E-5
50% Validity range not reached : 1.2471812624426093E-5
Integration duration performed so far : 4.999999873689376E-5
source I of circuit solvers loops : 0.0013742401 A.R
circuit integration - last compute in loop 0 : 1.0
circuit integration - init in loop 1 : 213.0
Nombre iterations CGS : 82.0 - 1671.0
Erreur (norme du residu) : 1.3157478E-17
Erreur (residu relatif) : 9.893993E-7
circuit integration - Linear solver in loop 1 : 62.0
Ratio of validity : 2.4941717128967866E-5
50% Validity range not reached : 2.4941717128967866E-5
Integration duration performed so far : 9.999999747378752E-5
source I of circuit solvers loops : 0.0013740167 A.R
circuit integration - last compute in loop 1 : 1.0
circuit integration - finalization : 0.0
end of circuit integration, iterative CGS method with tolerance = 1.0E-6 (total compute duration) : 3660.0
```

LinSolveDoublePrecision tol 1e-3:

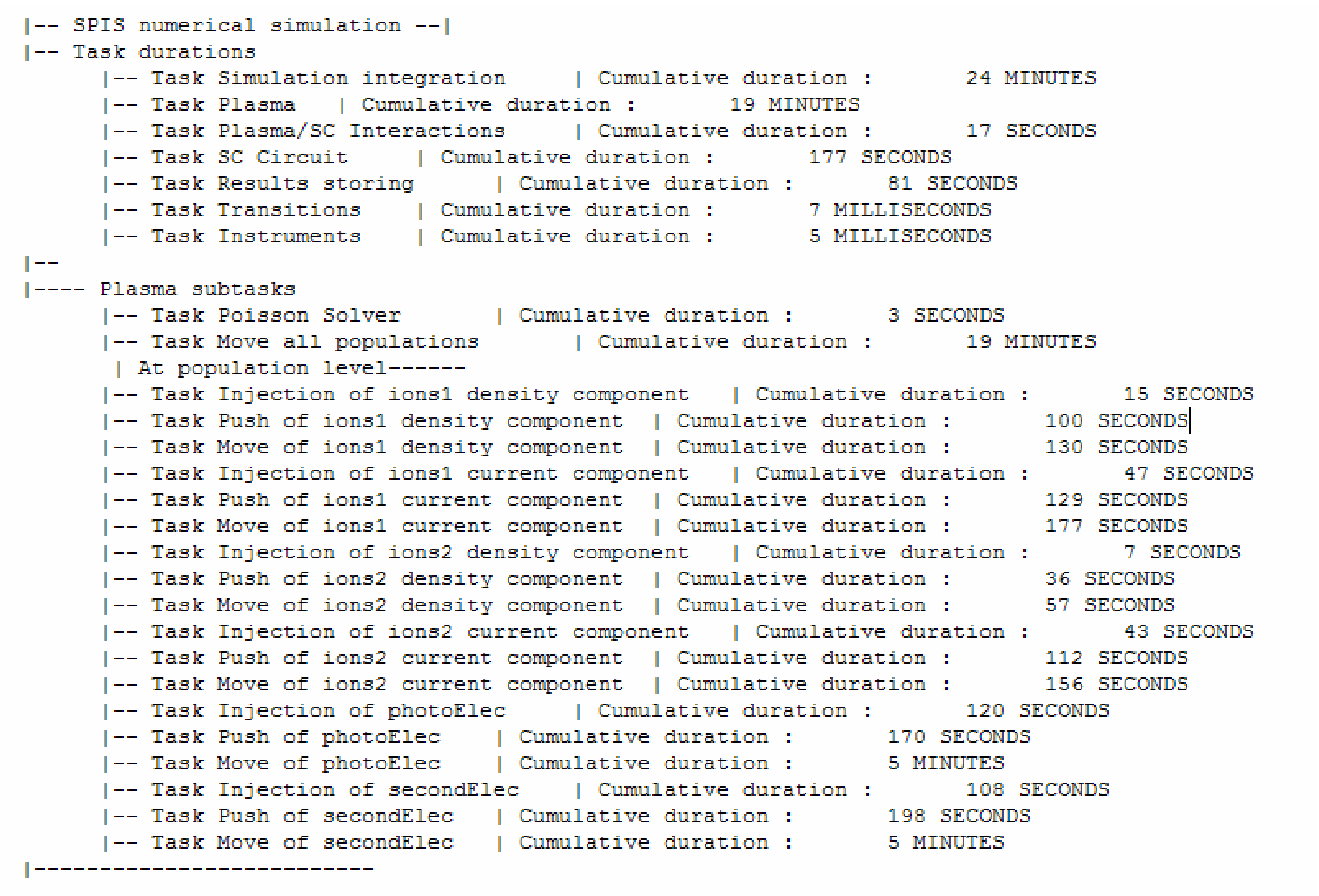


```
circuit integration start :
circuit integration - end of init : 3106.0
circuit integration - init in loop 0 : 198.0
Nombre iterations CGS : 43.0 - 1671.0
Erreur (norme du residu) : 5.860507E-15
Erreur (residu relatif) : 7.3916296E-4
circuit integration - Linear solver in loop 0 : 88.0
Ratio of validity : 1.2471812624426093E-5
50% Validity range not reached : 1.2471812624426093E-5
Integration duration performed so far : 4.999999873689376E-5
source I of circuit solvers loops : 0.0013742401 A.R
circuit integration - last compute in loop 0 : 1.0
circuit integration - init in loop 1 : 150.0
Nombre iterations CGS : 43.0 - 1671.0
Erreur (norme du residu) : 1.5097993E-14
Erreur (residu relatif) : 9.171182E-4
circuit integration - Linear solver in loop 1 : 61.0
Ratio of validity : 2.4941717128967866E-5
50% Validity range not reached : 2.4941717128967866E-5
Integration duration performed so far : 9.999999747378752E-5
source I of circuit solvers loops : 0.0013740167 A.R
circuit integration - last compute in loop 1 : 2.0
circuit integration - finalization : 0.0
end of circuit integration, iterative CGS method with tolerance = 0.0010 (total compute duration) : 3607.0
```

Figure 52 - CGS circuit solver efficiency compared to Gauss method

3.9.5. *Numerical monitors capabilities*

Monitoring of CPU time:
A new class of objects(*TaskComputingDuration*) has been created to monitor the CPU time of the most costly computing processes of the NUM core. As shown in the example below, the detailed CPU times is presented at the end of the simulation from top to low levels. Extra data (monitoring during the simulation) can be achieved via a new global parameter (*taskDurationVerbose*). This new feature is very useful to determine which numerical routines are the more costly and can help users to tune the numerical parameters.



```
|-- SPIS numerical simulation --|
|-- Task durations
      |-- Task Simulation integration     | Cumulative duration :       24 MINUTES
      |-- Task Plasma    | Cumulative duration :       19 MINUTES
      |-- Task Plasma/SC Interactions     | Cumulative duration :       17 SECONDS
      |-- Task SC Circuit     | Cumulative duration :       177 SECONDS
      |-- Task Results storing       | Cumulative duration :       81 SECONDS
      |-- Task Transitions     | Cumulative duration :       7 MILLISECONDS
      |-- Task Instruments     | Cumulative duration :       5 MILLISECONDS
|--
|---- Plasma subtasks
      |-- Task Poisson Solver       | Cumulative duration :       3 SECONDS
      |-- Task Move all populations       | Cumulative duration :       19 MINUTES
       | At population level------
      |-- Task Injection of ions1 density component    | Cumulative duration :       15 SECONDS
      |-- Task Push of ions1 density component    | Cumulative duration :       100 SECONDS
      |-- Task Move of ions1 density component    | Cumulative duration :       130 SECONDS
      |-- Task Injection of ions1 current component    | Cumulative duration :       47 SECONDS
      |-- Task Push of ions1 current component    | Cumulative duration :       129 SECONDS
      |-- Task Move of ions1 current component    | Cumulative duration :       177 SECONDS
      |-- Task Injection of ions2 density component    | Cumulative duration :       7 SECONDS
      |-- Task Push of ions2 density component    | Cumulative duration :       36 SECONDS
      |-- Task Move of ions2 density component    | Cumulative duration :       57 SECONDS
      |-- Task Injection of ions2 current component    | Cumulative duration :       43 SECONDS
      |-- Task Push of ions2 current component    | Cumulative duration :       112 SECONDS
      |-- Task Move of ions2 current component    | Cumulative duration :       156 SECONDS
      |-- Task Injection of photoElec     | Cumulative duration :       120 SECONDS
      |-- Task Push of photoElec     | Cumulative duration :       170 SECONDS
      |-- Task Move of photoElec     | Cumulative duration :       5 MINUTES
      |-- Task Injection of secondElec     | Cumulative duration :       108 SECONDS
      |-- Task Push of secondElec     | Cumulative duration :       198 SECONDS
      |-- Task Move of secondElec     | Cumulative duration :       5 MINUTES
|--------------------------
```

Plasma sensors:
As described in section 3.2.6, lots of plasma sensors can produce data along the simulation, including local monitors embedded in the computational volume. Most importantly is the possibility to add such instruments within the course of the simulation or to modify their parameters (position, frequency …). The numerical behaviour can be checked by users and increase the level of confidence one may have on the results obtained with the tool.

Illustrations:
An example of plasma sensor corresponds to the spinning sphere described in paragraph 3.7.4, Figure 42. Four sensors are used: 2 ion density sensors and 2 potential sensors. The oscillatory regime can easily be verified on next figures. A possible output for the user is an help to choose the densification to be applied to super particle number.

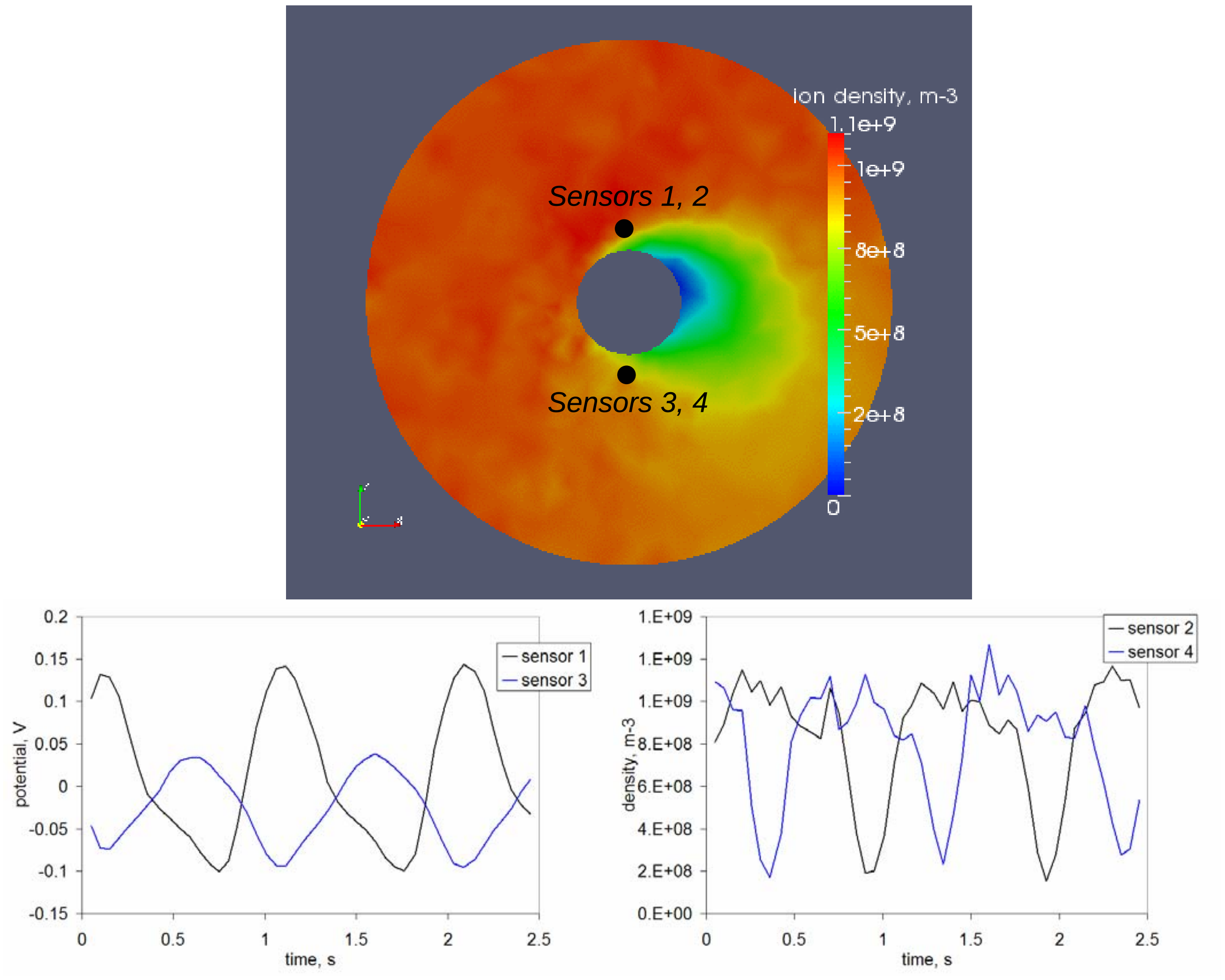


Figure 53 - Illustration of a possible use of plasma sensors.

3.10. Other developments

3.10.1. Spacecraft area

The total area of the spacecraft is now available in the log console at the initialization of the simulation. It aims at providing the real surface meshed. For example, depending on the mesh refinement, we can obtain differences of more than 2 % for a spherical probe.

3.10.2. Pause simulation

A volatile boolean variable *isPaused* has been added to *SimulationFromUIParams* ("volatile" means that can be modified by another thread, here SPIS-UI). A public accessor is given and UI provides the user with the possibility to click on a "pause" button" which automatically makes the numerical kernel providing the data at the current time step and wait the user giving order to continue the simulation with another total duration.

3.10.3. Control of simulationDt

The automatic increase of the simulation time step can be moderated by a new global parameter *simulationDtMaxFactor*. The maximal time step dt^{n+1} of the spacecraft loop n+1 is $dt^n \times$ *simulationDtMaxFactor*. In some sensitive cases, it can permit to perform a smoother time integration.

3.10.4. Charge deposit in volume

In previous version, charge deposit was computed at the end of particle trajectory integration from UI. However, numerical routines were already implemented for the deposit of the charge along the trajectory. It aimed at increasing the statistics since one particle could deposit its charge over a large number of cells. In some cases, it is clearly a advantage but in other cases, it can produce numerical instabilities of the Poisson-Vlasov system. It indeed can provoke large instable structures to develop within the volume. The work has then consisted in implementing a new global parameter *chargeDepositDuringIntegrationFlag* :

- *chargeDepositDuringIntegrationFlag* = 0 (default value) then no charge deposit during integration
- *chargeDepositDuringIntegrationFlag* = 1 then charge deposit during integration

It is the responsibility of the user to select this mode.

3.10.5. **Backtracking** *volume distribution*

The accuracy of the particle backtracking in *BackTrackingVolDistrib* class relies on the discretization of the maxwellian energy distribution used for injecting particles. The work has consisted in providing new global parameters for the geometric series describing the distribution:

- *maxwellEnergySamplerPointNb* : number of points of the series (default = 20)
- *maxwellEnergySamplerSpacing*: first relative spacing of the series (absolute value is obtained multiplying by the population temperature) (default = 0.02)
- *maxwellEnergySamplerFactor*: ratio of the geometric increase (default = 1.3)

NB: The current collection by a spherical probe in medium Debye length regime has been enhanced using more points in the discretization (error decreased from 1.5 % to less than 1.0%).