SPIS–UI, a NEW INTEGRATED MODELLING ENVIRONMENT for SPACE APPLCIATIONS

Julien Forest[1], Jean-Francois Roussel[2], Alain Hilgers[3], Benoit thiebault[3], Sebastien Jourdain[1]
(1) Artenum Company, Paris, France, (2) ONERA, Toulouse, France, (4) ESA/TEC, Netherlands

Introduction: the objectives of the SPIS project

The main objective of the SPIS project is to offer an open source, homogenous and extensible range of numerical tools to model precisely spacecraft plasma interactions. Regarding the spacecraft charging process, the scope of expected applications of SPIS is much larger with, for instance, the modelling of electrostatic sheath structure or the characterisation of space plasma instruments. New technologies like the electrical propulsion or high power solar generators open new technical issues and challenges, achievable only through an accurate modelling. Today, application domains are both scientific and industrial, taking into account different approaches, constraints and cultures of users. SPIS aims to be an easy-to-use and fast engineering and scientific code, accurate and adaptable to specific and complex cases.

From the User Interface (UI) point of view, to bring together both the scientific and industrial spheres represents a real challenge because their user requirements are different. The necessity to take into account a very complex modelling, that includes complex pre and post-processing phases, worsens this situation. The adaptability and extensibility, which are the key requirements, of the system prevent a classical approach based on monolithic codes. In order to follow the adaptability and extensibility of the framework, the user interface has to follow the Object Oriented Approach (OOA), and the possibilities of dynamic loading offered by modern languages, like Java and Python/Jython. Regarding the simulation kernel, SPIS-NUM, which constitutes the core of SPIS system, this analysis led to the development of a proper sub-project, called SPIS-UI, to handle the issues previously mentioned.

Specification identification method

A precise and clear identification of all specifications, and especially user requirements, is critical before the development of a complex and tailored user interface. User and software requirements have been collected by interaction with the SPINE community and have been gathered into two reference documents. However and in a continuous development approach, the core team continue to take into account the inputs and feedbacks of the community through the SPINE's Virtual Lab server. The specific phase is not closed and user requirements are regularly updated on this basis. The requirement collection has been built also on the basis of the previous experiences and charging modelling software, like NASCAP, Silecs, GEANT-4 or PicUp3D, and the culture and habits that they have already built. These tailored experiences have been completed by the analysis of more generic tools like the CERN/ROOT and the SALOME framework, Integrated Development Environments (IDE) like Eclipse or light prototyping/analysis tools like Matlab.

The modelling chain and the need of an advanced User Interface

As illustrated Figure.1 the modelling of a complete spacecraft-plasma system is a complex demarche. During the pre-processing phase, the system should be built before any simulation. This includes, at least, the CAD modelling of the spacecraft geometry, the definition of the boundary condition and the setting of the characteristics of the plasma environment. If the mode is based on an unstructured mesh like SPIS, the pre-processing phase should also include the meshing of the computational volume. At the issued of the pre-processing phase, the system should be completely built and ready to be inject as input into the solver. At the end of the chain, during the post-processing phase, results of the simulation should be analysed and processed by generic and specific treatments, based on the experience of the user.

In this context, the simulation run itself seems appear as only a simple task of a multi-step modelling chain, that the user interface should handle in its globality.

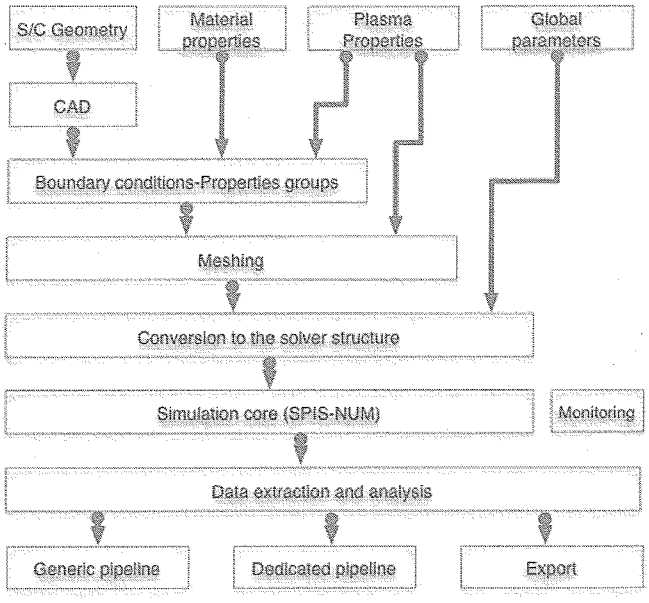

Figure 1: Modelling chain

Tasks of modelling chain are not always performed in the same order and the possibility of iterations and partial sub-cycles is requested. The user interface should help the user to respect this chain, built a self-consistent system, drive the simulation and extract and analyse his results. Behind the user interface itself, the main difficulty is the conversion of the various data structures between each tasks, maintenance of their consistency. Traditionally, this chain is performed using many independent and standalone tools, the link between each element being done by a set of one-to-one conversion scripts. This approach is quickly limited. On the other hand, monolithic codes are rigid and difficult or too costly to adapt, especially if the user himself has to modify them. The requirement of software in the same time easy to use and an adaptable and extensible is then very difficult to respect.

Collected users requirements show the manners how to use the software may strongly differ from one domain of application to another one. And one critical aspect is that many levels of interaction are needed. The software must be controllable from an advanced graphic user interface (GUI). But the user should also have the possibility to access directly to the source code, in order to adapt them to specific cases. However, the learning curve might be here quite stiff for new users. In parallel, a lighter access with a high level script language is then useful, especially to perform direct and rapid prototyping, and must be introduced.

Concept and design

The concept of framework can handle this issue. The framework approach is based on a structured integration of independent tools, under the form of modules put together through an open platform of integration. Actions on modules are controlled through a central manager, responsible in parallel to check the consistency of data. Data exchanges between modules are performed thought a common exchange bus and a set of common data structures. In this approach, data must be converted only between each module and the common data structure only. The progressive extension and adaptation are then much easier, just by integration of additional modules into the framework. Platforms like ROOT [1] or SALOME [1] has open up this approach, that been used and extended in SPIS-UI.

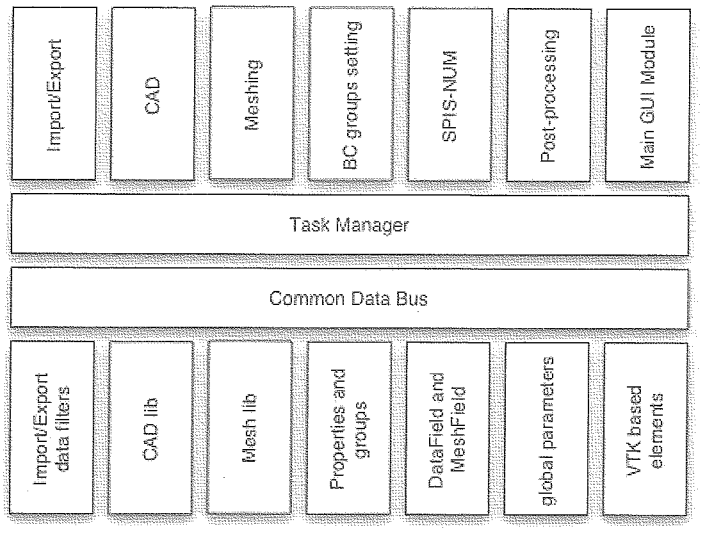

Figure 2: Top design of SPIS-UI

Figure.2 presents the top design of the SPSI-UI framework. Each step along the modelling chain is considered as a unit and generic *Tasks*, in which ones external software are wrapped or integrated as simple plug-ins. A central *Task Manager* controls checks the dependencies between each task according to a dependence tree. When a task is called, the task manager checks if all task previously needed have been performed and, if not, executes them automatically. The basic configuration of SPIS-UI includes a CAD definition module, a meshing task and a set of post-processing tasks. The simulation kernel and the main GUI are themselves wrapped into generic tasks as well. The GUI layer is divided into two levels. A main GUI module controls the Task Manager and builds up the graphical context under the form of a central common desktop. Each task has its own GUI that can be projected and interfaced into the desktop. This approach authorises in parallel that most of the integrated modules can be used as well as independent and standalone software, independently of SPIS, or can be intimately integrated into the framework.

SPIS-UI includes four main common data structures to exchange data. A generic CAD library is used to describe the geometry of the spacecraft. A generic mesh library allows to stock in memory 2D/3D unstructured grids. The *dataField* and *meshField* structures are used to defined fields, scalar or vector, living on the mesh. Surface properties of the spacecraft and plasma characteristics are stocked in a *properties* structure. Various additional structures, like global parameters complete the system. All structures are referenced into a central dictionary and can be called by simple key words. The framework can be seen as a kind of "operational system" dedicated the modelling and where each numerical tool can be loaded, executed and interacts with other ones.

Technologies and methods

The intrinsic polymorphism of tasks and data imposes to use an object approach in the design of the common data structure. Extensions to new data or types of module can be done just by derivation of the initial common interface. The second issue is that we have to integrate very heterogeneous tools, written in various languages and based on incompliant architectures. This point is more difficult to solve, especially if we want keep an approach the lightest possible, in order that new users can go into the code easily without an extensive knowledge in computing science. For this reason, generic client-server approaches, like CORBA, has been avoided. One solution is to use "glue languages" and the wrapping techniques. The glue languages are high-level languages, generally script based and interpreted by a virtual machine, able to be interfaced with lower level languages. Native modules are then wrapped and can be called and controlled by a few instructions of script.

This approach was followed in SPIS-UI with the combination Jython as glue language and Java /C++ as low-level languages. Python is a powerful object oriented script language, having a very clear syntax, that facilities the code visibility and reduce the learning curve with respect other object oriented languages. It reduces the size of source code and the corresponding development time in a significant manner. Python is for example very efficient for quick prototyping. For these reasons, Python is more and more used in the industry in replacement of meta-languages like MatLab's one or to replace Pearl. Due to the choice to develop SPIS-NUM in Java, Jython, the java interpreted version of Python, was chosen as glue language in SPIS-UI. Both Python and Jython are open-source projects supported by

very dynamic communities coming from the open-source, scientific and industrial world. They are now stable and validated technologies. Jython presents a very intimate interface with respect to Java. All Java based object can be dynamically loaded and manipulated through Jython without specific treatment or setting. New objects can be hot plugged, dynamically instanced and manipulated. The Java Native Interface, JNI, is used to wrap native elements written in languages like C/C++. This technique is especially used for the integration the VTK library, used for the visual post-processing modules. In theory, the system can be extended to the integration of Fortran based subroutines with a C/C++ intermediate layer. However, this technique has never been tested in details. Figure.3 illustrates the various configurations of wrapping used in SPIS-UI to control and interface tasks.

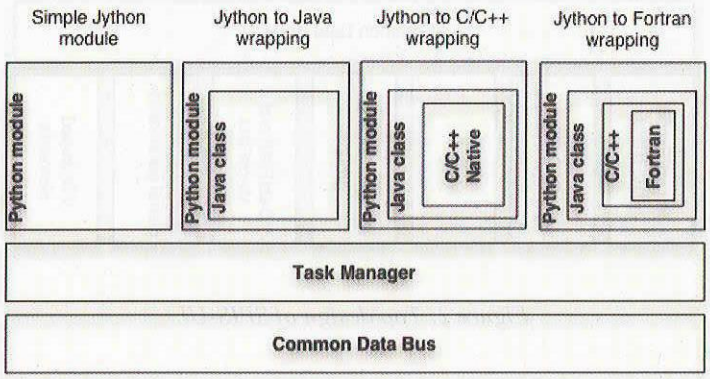

Figure 3: Integration of modules by multi-layer wrapping.

All wrappings are dynamic and all elements of SPIS can be edited, modified and reloaded into the framework without to restart this last one. This property changes deeply the way to approach the whole modelling chain, because models can be dynamically modified and applied on the basis of data already present on the framework. This is in particularly possible with the simulation kernel. Source codes of SPIS-NUM can be edited, re-compiled, reload into the framework and a new run performed. Some elements are simply executed as external executable codes and simply interfaced by import/export files.

SPIS-UI in practice

Figure.4 presents the corresponding result and show SPIS-UI in interactive mode with its graphic user interface and a few examples of loaded tasks. The main graphic interface gives the context for dedicated interfaces specific to each task. A Jython console allows the direct manipulation of the elements of the framework.

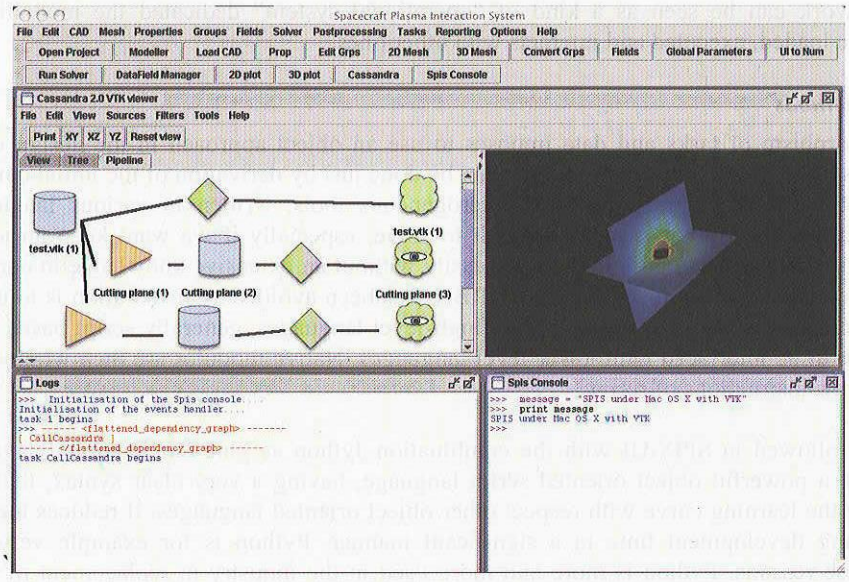

Figure 4: Main Graphic User interface of SPIS-UI

The CAD module is based on the open-source CAD editor and mesher Gmsh. For now, Gmsh is executed as simple external standalone executable software and its own GUI used for the modelling of the spacecraft geometry. The spacecraft geometry is described on the classic breps approach and is similar to other classic CAD tools, as illustrated in Figure.5.

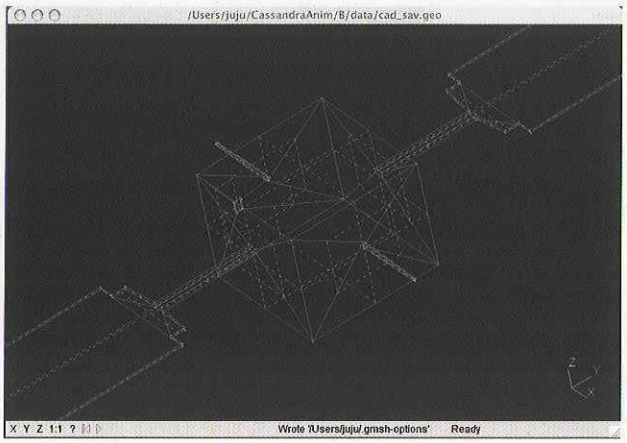

Figure 5: CAD modelling with gmsh.

Gmsh performs also the meshing of the computational volume and the spacecraft surface. A Group Manager allows to define groups of elements of CAD to attribute boundary conditions and fields defined locally on the grid. Different types of material properties are already defined in the same approach than NASCAP. Plasma and electrical properties can also defined and applied in the same manner. SPIS-UI manages the conversion of the groups of CAD to groups of mesh elements and the mapping of the corresponding fields on the grid. At the end of the pre-processing phase, all initialisation and boundary fields are mapped on the grid and the system reading to be loaded into the solver. An additional module may be used to set the global parameters.

Simulation control

According to the requirements, SPIS-UI offers at least three ways to execute and control the simulation. SPIS-NUM is wrapped as a Task through a Jython script. This script call SPIS-NUM on the basis of a set of predefined models, that the user can directly executed and controlled from the main GUI. If the pre-defined models are not adapted, the user can separately edit the source codes of SPIS-NUM, to define his own model, compile them and reload them into the framework. Last, the user can directly access to all objects of the SPIS-NUM structure through the Jython console and directly build his model dynamically. This approach is especially useful for the prototyping phase. An object-oriented completion of the command line is currently under validation to facilitate this approach.

Data Analysis and post-processing

Due to the increasing volume and complexity of data produced by modern simulation software, the analysis of result or post-processing is becoming more en more complex. The constraints and specifications of the post-processing chain are similar to constraints of the whole modelling chain. The post-processing tools must be, on one hand, easy to use with an advanced GUI and many pre-defined filters and treatments available, and, on the other hand, adaptable to specific cases, especially for 3D data.

All 3D visualisation module of SPIS-UI are based on the VTK [3]. VTK is a powerful open-source library of 2D/3D visualisaiton and processing. VTK is based a kernel written in C++ and offers an extensive set of filter and data analyser. Treatments are the notion of visualisation pipelines, where the data processed are transferred to a mapper, making the conversion into colour maps, to finally build an actor of representation. Filters can be piped to create new types of filters. VTK is supported bt a very dynamic community and is progressively becoming an open-standard for the scientific 3D visualisation. The learning curve of the VTK library is unfortunately stiff and its use requires a good knowledge of the object oriented programming in C++. In parallel of the SPIS project, Artenum has developed the CASSANDRA VTK based viewer. Figure.6 presents its graphic user interface.

Cassandra integrates a visualisation pipeline manager that allows to manage dynamically the visualisation pipelines. A graphical pipeline editor allows to build very easily and quickly all type of pipeline by combine of basic filters. Filters are external plug-ins dynamically re-loadable. A generic Java interface between the plug-ins and Cassandra simplifies strongly the development and the integration of new plug-ins. The set of available plug-ins is progressively extended.

In the same approach than SPIS, a Jython console completes the system. The user can directly manipulation of defined processing or define in an external Jython script complete new pipelines.

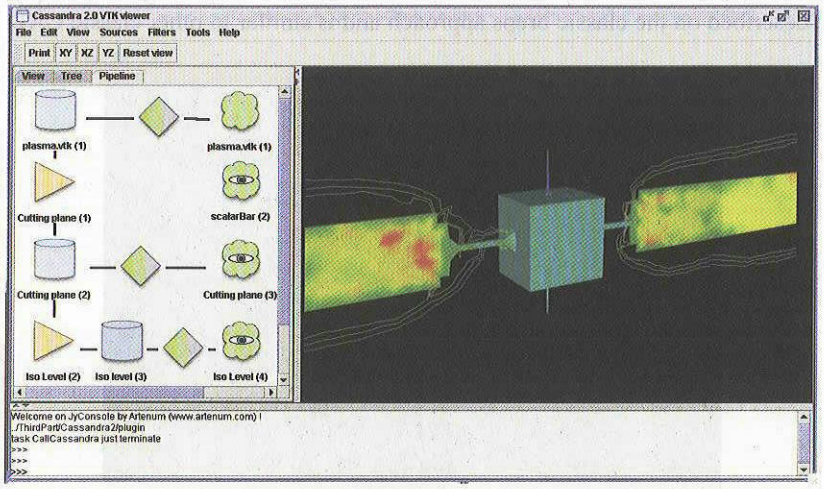

Figure 6: View Cassandra, the 3D viewer used in SPIS-UI.

Conclusion

SPIS-UI, the generic framework of SPIS, has been presented. The main accent of the current development has been put on the adaptability and extensibility of the framework and the possibility to offer many levels of interaction and control. SPIS-UI is still under validation and many features will be modified and extended in a near future. However, first tests and validations runs performed with SPIS have already opened a possible new approach in the modelling of such systems. SPIS-UI is first used to build the system to be modelled and generate all data needed to execute sub-routines of the numerical model or processing modules. On this basis, numerical and processing sub-routines can be directly developed and tested. The integration of an object oriented completion in the command line and advanced editors able to visualise the syntax of the Jython and Java languages complete the approach. The demarche tends to be close to the approach of IDEs like Eclipse. Another direction of extension of SPIS-UI is the improvement of its import/export modules and the integration of other tools of spacecraft plasma interaction modelling. For example, PicuUp3D is currently under integration. One possible objective is to make on the basis of SPSI-UI an open-standard to facilitate the interaction and the data exchange between spacecraft charging simulation codes. One of the most direct benefit of this strategy is the improvement of the confidences in the codes by direct cross-validation or the definition of common reference cases. In parallel, this opens the possibility of the creation of multi-physics models, by the integration of numerical kernels from various fields.

Acknowledgments

This work has been funded by ESA under contract No. 16806/02/NL/JA. We also acknowledge interesting discussions with SPINE community members, and their very useful testing of the code.

Refrences

[1] CERN, Root Project's Web page, http://root.cern.ch/, 1995

[2] Salome project's Web page, http://www.salome-platform.org/, 2003

[3] Kitware, VTK's Web page, http://public.kitware.com/VTK/, 2001

[4] Forest, J., "New Open Source Spacecraft Plasma Interactions Simulation Software Library, PicUp3D: First Tests and validations", submitted to *Journal of Spacecraft and Rockets*, in 2005.

[5] Hilgers, A. et al, " Tests and Validation of a New Spacecraft Plasma Interaction Software, SPIS", proceeding of 9th Spacecraft Charging Technology Conference, Tsukuba, Japan, April 4-8, 2005.

[6] Roussel, J.-F. "Spacecraft Plasma Environment and Contamination Simulation Code — Description and first Tests", *Journal of Spacecraft and Rockets*, Vol. 35, N. 2, 1998, pp 205-211.

[7] Roussel, J.-F., et al, "Design of a New Modular Spacecraft Plasma Interaction Modeling Software (SPIS)", *proceeding of 8th Spacecraft Charging Technology Conference*, Huntsville, AL, USA, Oct. 20-24, 2003.