Computational tools for spacecraft electrostatic cleanliness and payload accommodation analysis -

SPIS-SCIENCE

Interface Control Document (ICD)
Software Life Cycle (SLC)

June 2013

ESTEC Contract No. 4000102091/10/NL/AS

ARTENUM

prepared in partnership with
ONERA
Interface Control Document (ICD)
Software Life Cycle (SLC)

June 2013

Written by
Julien Forest
Benoît Thiébault
Benjamin Ruard

ARTENUM
24 rue Louis Blanc
75010 Paris
France

Verified by
Jean-Charles Matéo-Vélez
Pierre Sarraillh

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

Document Status Sheet

<table>
<thead>
<tr>
<th>Issue</th>
<th>Date</th>
<th>Author(s) of document/change</th>
<th>Reason of change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>14/06/2013</td>
<td>JF</td>
<td>Creation</td>
</tr>
</tbody>
</table>
TABLE DES MATIERES

1. INTRODUCTION.......................................................................................................................... 4
   1.1. Objectives ................................................................................................................................. 4
   1.2. Acronyms .................................................................................................................................. 4
   1.3. Applicable Documents.............................................................................................................. 5
   1.4. Reference Documents ............................................................................................................. 5

2. REVIEW OF EXTERNAL COMPONENTS AND INTEGRATION SCHEMES .................................. 6
   2.1. SPIS general design and external component relation-ship...................................................... 6
   2.2. External components ................................................................................................................ 9
   2.3. Software life-cycle, compilation process and configuration control ........................................ 13
       2.3.1. Compilation and installation process ............................................................................... 14
       2.3.2. Centralisation and access to the sources codes ............................................................. 15
1. INTRODUCTION

1.1. Objectives

This document identifies and describes external components and technologies used in SPIS and presents the integration and inter-dependencies schemes of these elements. Last the normalized compilation process, based on the Maven-Hudson-Sonar continuous chain is presented.

1.2. Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>Architecture Design Document</td>
</tr>
<tr>
<td>BT</td>
<td>BackTracking</td>
</tr>
<tr>
<td>ESN</td>
<td>Electrical Super Node</td>
</tr>
<tr>
<td>FT</td>
<td>ForWardTracking</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>IV</td>
<td>Current voltage sweep</td>
</tr>
<tr>
<td>LP</td>
<td>Langmuir Probe</td>
</tr>
<tr>
<td>NUM</td>
<td>Numerical core of SPIS</td>
</tr>
<tr>
<td>ONERA</td>
<td>Office National d'Etudes et de Recherches Aérospatiales</td>
</tr>
<tr>
<td>PD</td>
<td>Particle detectors</td>
</tr>
<tr>
<td>S/C</td>
<td>Space craft</td>
</tr>
<tr>
<td>SPIS</td>
<td>Spacecraft Plasma Interaction Software</td>
</tr>
<tr>
<td>SPIS-CORE</td>
<td>Current SPIS development branch available on spis.org website</td>
</tr>
<tr>
<td>SPIS-GEO</td>
<td>Simplified MEO/GEO tools for spacecraft charging, ESA Co4000101174</td>
</tr>
<tr>
<td>SPIS-SCI</td>
<td>SPIS-SCIENCE: Computational tools for spacecraft electrostatic cleanliness and payload accomodation analysis</td>
</tr>
<tr>
<td>STG</td>
<td>Semi-Transparent Grid</td>
</tr>
<tr>
<td>TP</td>
<td>Test Particle</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface of SPIS</td>
</tr>
<tr>
<td>UR</td>
<td>User Requirement</td>
</tr>
<tr>
<td>PS</td>
<td>Plasma Sensor</td>
</tr>
<tr>
<td>SRD</td>
<td>Software Requirements Document</td>
</tr>
<tr>
<td>VPD</td>
<td>Virtual Particle Detector</td>
</tr>
<tr>
<td>wrt</td>
<td>with respect to</td>
</tr>
<tr>
<td>IME</td>
<td>Integrated Modelling Environment</td>
</tr>
</tbody>
</table>

---
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.


1.4. Reference Documents

[SPINE WS] 17th SPINE meeting (documents, presentations) :  
http://dev.spis.org/projects/spine/home/meeting/mxvii


[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. REVIEW OF EXTERNAL COMPONENTS AND INTEGRATION SCHEMES

2.1. SPIS general design and external component relationship

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 (controller), Keridwen is designed as a modular framework where each functional block is dynamically integrated under the form of normalised OSGI bundles. In the packaged releases, bundles stored in the `lib` and `dependencies/main` sub-directories of the Felix package are loaded at the runtime. By default, the selection of bundles being copied in these directories is done at the packaging phase but can be modified manually without recompilation of the whole framework. The loading order is automatically computed by Felix in function of the interdependencies (in term of active services) of OSGi modules. One should outline that the fact that bundles are loaded, and available as services, should be distinguished from the wizard-based scenarios performing the whole modelling process and presenting the various GUI in a specific order. These ones are defined through XML based files.

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 gathers 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 1, hereafter, 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 1: Interaction tree between SPIS related modules and external components. Transitive dependencies are not illustrated here.
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.

The various SPIS modules and their relationship with external modules are described in the ADD/SDD document. Please refer to this document for further information.

For information, regarding the Keridwen IME, on which one SPIS-UI is based, the main modules are:

- **org-keridwen-core-data-model**: Define the interface generic and central data-model. Dedicated concrete implementations for SPIS have been especially done in the SPIS developments;
- **org-keridwen-core-i18n**: Module of management of the internationalisation of the UI;
- **org-keridwen-core-messaging**: Central message-driven controller;
- **org-keridwen-core-logging**: Generic logging module;
- **org-keridwen-core-settings**: Generic setting module;
- **org-keridwen-core-state**: Central state machine for the process control (i.e. data-wizard consistency process);
- **org-keridwen-core-ui-dockingframe**: Generic interface with the Docking Frame GUI framework;
- **org-keridwen-gmsh**: Generic interface with the Gmsh CAD/mesh tool;
- **org-keridwen-modelling-console**: General message and script consoles;
- **org-keridwen-modelling-global-parameters**: Global parameter editor;
- **org-keridwen-group-editor**: Generic properties and groups editor;
- **org-keridwen-modelling-reporting**: Generic autop-reporting module;
- **org-keridwen-text-editor**: Generic text file editor;
- **org-keridwen-utils**: Various utilities;
- **org-keridwen-vtk**: Interface with VTK based components;
- **org-keridwen-xstream**: Interface / packaging of the Xstream module;

Figure 3 shows the inter-dependencies between Keridwen and other external components.

We remind that the whole compilation process of SPIS and its sub-components is based on Apache Maven. This tool manages by its own the needed dependencies during the compilation, through normalised Maven artifacts. All of these ones are stored on online Maven repositories. To access to them an access to Internet is required to download dependencies properly. Figure 2 shows an example of Maven repository. Artifacts corresponding to Artenum’s products are stored on the public Artenum’s Maven repository (see maven.artenum.com). The license itself is generally given in the LICENSE.txt file at the root path.
Figure 2: View of the Artenum’s Maven repository. Illustrated here the artifact of the SPIS-UI OSGi bundle for the electrical circuit editor. If this bundle is required in one development, the developer has only to copy-past the XML artifact (right-down corner) into its own pom.xml file.

2.2. External components

We remind that SPIS integrates and is based on several external software components (or third party components). All external components are integrated according the terms of open-source licenses (e.g. GPL, LGPL, BSD, Apache,...). Some of them are directly linked to the present development (i.e. import) or dynamically at the runtime (i.e. OSGi bundles) or as external tools through a system call (e.g. mesher).

We remind that all of these components are compiled using Apache Maven and the corresponding license is indicated in the artifact description in the Maven configuration files (i.e. pom.xml) and, in most of the cases, given in the LICENSE.txt file. More generally, licenses can be automatically generated with the standard following Maven command:

```
mvn site
```

This last command generates locally a complete html site, gathering all key information related to the present source codes (description, licenses, API documentation, dependencies). Please notice that the locally
generated sites are generated module by module and cross-links between modules can be still broken until the sites are deployed on the official and public server.

The components listed above are directly used by SPIS but require the following technologies:

Regarding the numerical core, SPIS-NUM, uses the same external components libraries as before:
- **Lapack netlib library**: LAPACK is a freely-available software package. It is available from netlib via anonymous ftp and the World Wide Web at http://www.netlib.org/lapack. Thus, it can be included in commercial software packages (and has been). It is only asked that proper credit be given to the authors. The license used for the software is the modified BSD license, see:
- **Blas netlib library**: Is released under the same conditions than Lapack.

Regarding the SPIS-UI IME, the following external components are used, as direct dependency:
- **Artenum’s Keridwen Core** corresponds to the kernel and low level components of Keridwen and is released under the GPL license (see www.kerdiwen.org);
- **Artenum’s Keriden Tools** corresponds to the generic top level functional modules of Keridwen and is released under the GPL license (see www.kerdiwen.org);
- **Artenum’s Penelope** is a generic 3D unstructured mesh developed in Java and is released under the GPL license. This library replaces JFreeMesh of the previous versions of SPIS. Penelope takes in charge the mesh memory structure and all types of data fields deployed on the mesh. Penelope also perform their conversion from/toward various formats (NetCDF, VTK, Gmsh...);
- **Artenum’s Frida** is a generic properties definition, storing and deployment library. Frida allows defining generic properties, like material, numerical properties, local settings, through theirs characteristics (e.g conductivity, color, temperature...). In connexion with Penelope, Frida allows to deployed such characteristics as corresponding DataFields on meshes. Frida is typically used in the Group Editor to define and apply material properties and pre-processings local parameters. Frida is released under the GPL license (see dev.artenum.com/frida).
- **Apache Felix Runtime** is the default OSGI runtime used to run the whole application. Felix Apache is available at http://felix.apache.org and is released under the Apache license V2.0.
- **VTK** is the 3D visualisation and data analysis library used for all 3D rendering and post-processing. VTK is a native component. A compiled version is provided for each supported platform on the Artenum’s Maven repository (see maven.artenum.com). The provided packaging includes the OSGI embedding. VTK is an open-source toolkit licensed under the BSD license.
- **Gmsh** is the CAD and meshing tool used in the pre-processing phase to define the studied geometrical system and perform the meshing of the computational domains and related elements (e.g surfaces meshes). By defaults, Gmsh is run through a system call, but a Java/JNI wrapping has also be developed by Artenum on its own effort and offered to the Gmsh community. A compiled version is provided for each supported platform. The default version of Gmsh includes the OpenCascad based plugin allowing to load STEP based CAD files. Gmsh is copyright (C) 1997-2013 by C. Geuzaine and J.-F. Remacle and is distributed under the terms of the GNU General Public License (GPL) (Version 2 or later, with an exception to allow for easier linking with external libraries).
**Artenum’s Cassandra** is the generic 3D visualisation and data analysis application, based on VTK, used for all 3D renderings and post-processing functions. Cassandra is released under the QPL license (see dev.artenum.com/cassandra).

**Artenum’s JyConsole** is the generic Jython/Pyton console provided to run and edit in interactive mode scripts based commands in SPIS-UI. JyConsole is released under the QPL license (see dev.artenum.com/jyconsole).

**Artenum’s ArtTk** is a Java/Swing based library offering an extensive toolkit for scientific and technical applications. ArtTk is available at the following address http://dev.artenum.com/projects/ArtTk and is released under the LGPL license.

This list of direct dependencies is completed by the following components, as indirect dependencies (transitive). Please see the relevant Web sites for further information about these elements (default JVM and Felix dependencies not listed):

- **Apache Commons Beanutils** (see http://commons.apache.org/proper/commons-beanutils). It is released under the Apache license V2.0.
- **Apache Commons Digester** (http://commons.apache.org/proper/commons-digester/), it is released under the Apache license V2.0.
- **Apache commons** (http://commons.apache.org/proper/commons-jexl/), it is released under the Apache license V2.0.
- **Apache Felix Ipojo** (http://felix.apache.org/site/apache-felix-ipojo.html), it is released under the Apache license V2.0.
- **Apache Xalan** (http://xalan.apache.org/), it is released under the Apache license V2.0.
- **Jython** (http://www.jython.org/) is available at the following address www.jython.org and released under the Python software foundation license V2.
- **Apache Xerces** (http://xerces.apache.org/), it is released under the Apache license V2.0.
- **Logback Project** (http://logback.qos.ch/), it is released under the Apache license V2.0.
- **Apache XML Project** (http://xml.apache.org/), it is released under the Apache license V2.0.
- **Slf4j Project** (http://www.slf4j.org/), it is released under the Apache license V2.0.
- **Apache Log4j Project** (http://logging.apache.org/log4j/1.2/), it is released under the Apache license V2.0.
- **Apache Commons Collection** (http://commons.apache.org/proper/commons-collections/), it is released under the Apache license V2.0.
- **Apache Commons IO** (http://commons.apache.org/proper/commons-io/), it is released under the Apache license V2.0.
- **Apache Commons Logging** (http://commons.apache.org/proper/logging/), it is released under the Apache license V2.0.
- **Xom Data Model** (http://www.xom.nu/), XOM is a new XML object model. It is an open source (LGPL), tree-based API for processing XML with Java.
- **Xstream** (http://xstream.codehaus.org/) is a simple library to serialize objects to XML and back again. XStream is open source software, made available under a BSD license.

Each of these components is provided under the form a standardised OSGI bundle and are directly provided with the packaged version of Felix given in the SPIS releases.
Figure 3: Inter-dependencies scheme between Artenum's components used in SPIS-developments. Cycles are avoided.

Please notice that due to the module design the SPIS-UI IME, news modules can be added in the further evolutions and the dependence tree extended. In addition, due to the dynamic loading mechanisms, modules are loaded at the runtime. And the compilation and the runtime dependence trees may differ.

Please notice that for each module the complete dependency tree can be automatically with the standard following Maven command:

```
mvn dependency:tree
```
In the packaged SPIS releases, the outputs are gathered into the Thirdparties.txt files at the root directory of the releases.

2.3. Software life-cycle, compilation process and configuration control

In order to both keep the most clear and detailed tracking of sources codes and components, on one hand, and simplify the compilation and packaging process, the whole software life-cycle has been fully reviewed in the frame of SPIS-GEO/MEO according the current industrial state-of-art. A continuous integration chain, based on Hudson, Junit and Sonar completes now the existing LibreSource hosting facilities (ww.spis.org).

The whole compilation process is now controlled through Apache Maven in order to, first, handle automatically all needed dependencies (jars and versions) in a stable manner and, second, allows automatic remote building, testing and packaging on the Hudson server (see ci.artenum.com).

In this approach, source codes must follow a strict and normalised structure (see Maven recommendations) and an automatic building and non-regression tests procedure are automatically performed every night.

Figure 4: View of the Hudson compilation job form SPIS-UI. Same configuration is done for SPIS-NUM.
Source code metrics and quality can be analysed and displayed through the Sonar server (see sonar.artenum.com).

![Sonar dashboard image]

Figure 5: Example of source code metric using Sonar: This treemap presents the respect of programming rules according to the Oracle/Sun recommendations for the Java language. The respect of such rules not only illustrates the source code quality but also indicate its maintainability and extensibility. Greenest is the best.

2.3.1. Compilation and installation process

The compilation process is normalised for each modules and centralised at the root directory of the SPIS-UI development root directory. The compilation of the whole application can be done just by typing the usual command:

```
mvn --DskipTests=true clean install
```

Generated binary files are stored in the local target and .m2 directories.

To run the compiled modules, the developer should download a pre-set Felix runtime (see below). Due to their specific packaging and/or their OS dependencies, some external modules should be recovered and installed manually:

- Xstream modules
• VTK native components.

Both are available on the Artenum’s Maven repository (see maven.artenum.com).

In the Felix installation, copy the $FELIX_ROOT_DIR/build-config/scripts/LOCAL_PLATFORM/org-spis-science locally, edit it in order to adapt the path to each modules to your local .m2 directory.

Run the runtime with the command:

./SpisLOCAL_PLATFORM.sh

When the command line of the Felix console appears, type the following command:

source ./org-spis-science

The application should start in graphical and interactive mode.

2.3.2. Centralisation and access to the sources codes

The version control system for developed source has been move to Subversion for reason of normalisation and integration into the continuous integration chain.

SPIS related developments are available on the SPINE platform (www.spis.org) at the following addresses:

• **SPIS-NUM** (http://dev.spis.org/ls-svn/SpisNum/branches/SPIS-SCI-GEO), this Subversion repository contains all sources codes related to the SPIS-NUM numerical kernel.
• **SPIS-UI** (http://dev.spis.org/ls-svn/SpisUI), this Subversion repository contains all sources codes related to the SPIS-UI IME.
• **Felix** (http://dev.spis.org/ls-svn/SvnFelixSpis), this Subversion repository contains a preset Felix Server, including all needed dependencies and setting files, to run SPIS.
• **SPIS Instruments** (http://dev.spis.org/ls-svn/SpisInstruments), this Subversion repository contains all developments related to the SPIS Instruments. These developments have been in the frame of the present activities.

External components having updated/modified in the frame of the general evolutions of SPIS are available at the following addresses:

• **Keridwen Core**: http://dev.artenum.com/ls-svn/keridwen
• **ArtTK**: http://dev.artenum.com/ls-svn/ArtTK
• **Nisaba**: http://dev.artenum.com/ls-svn/nisaba
• **Cassandra**: http://dev.artenum.com/ls-svn/Cassandra
• **Elegance**: http://dev.artenum.com/ls-svn/eleganceSync2
• **Penelope**: http://dev.artenum.com/ls-svn/PenelopeProject/Penelope
• **Frida**: http://dev.artenum.com/ls-svn/Frida
• **Keridwen Tools**: http://dev.artenum.com/ls-svn/keridwen-tools