Deliverable #4

Architecture Design Document

	Name	Date
Prepared by	Benoît Thiébault, Project Manager	04/11/2010
Checked by	Jérémie Turbet, Software developer	08/11/2010
Approved by	Julien Forest, Expert Consultant	09/11/2010

Reference: ESA-SPISGEO-D4-ADD-2010-11-002	**Version**: 1	**Revision**: 0
Contract number: 4000101174	**Date**: 04/11/2010	

Table of contents

Summary

Objectives

The purpose of this document is to define a collection of software components and their interfaces to establish a framework for developing the software.

Technical overview

SPIS-GEO application is proposed to be built on top of Keridwen 2.0, an OSGi-based Integrated Modelling Environment to achieve a modular application, easier to maintain and to integrate with other tools.

A description of the components of this modular application is given as well as the compliance matrix with the software requirements.

Diffusion

Nom	Organisation
David Rodgers Alain Hilgers Fabrice Cipriani	ESA
Julien Forest Benoît Thiébault Jérémie Turbet	Artenum
Jean-Charles Matéo Vélez Jean-François Roussel Pierre Sarrailh	ONERA
Bjarne Andersson Alain Demairé	SSC
Patrice Pelissou Marc Sevoz	EADS-Astrium

Changes record

Version	Revision	Date	Author / Modification
1	0	04/11/2010	Benoît Thiébault / Document creation

1. Introduction

1.1. Scope of the document

The purpose of this document is to define a collection of software components and their interfaces to establish a framework for developing SPIS-GEO.

1.2. Reference and applicable documents

1.2.1. Applicable documents

[AD1] Statement Of Work "Simplified MEO/GEO Tools for Spacecraft Charging", TEC-EES/2008.348/DR, issue 1.0, 30/09/2009.

[AD2] Technical and Administrative Proposal, ESA-SPISGEO-PTC-2009-12-001, 20/11/2009

[AD3] Software Requirements Document, ESA-SPISGEO-D3-SRD-2010-11-001.

[AD4] Guide to applying the ESA software engineering standards to small software projects, BSSC(96)2 Issue 1, 1996.

[AD5] Guide to the software architectural design phase, ESA PSS-05-04 Issue 1 Revision 1, March 1995

1.2.2. Reference documents

N/A

1.3. Acronyms and abbreviations

- **GUI**: Graphical User Interface
- **S/C**: Spacecraft
- **SPIS-CORE**: Current Spacecraft Plasma Interactions Software main development branch that is available on spis.org website (version 4.2).
- **SPIS-GEO**: Simplified Standard MEO/GEO Tools for Spacecraft Charging
- **TBD**: To Be Determined
- **UR**: User Requirement
- **WC**: Worst Case
- **W.R.T.:** with respect to

2. System overview

2.1. Background and context

Initially released in March 2004, the SPIS software has become today the European standard for the modelling and the simulation of the spacecraft plasma interactions. Initially funded on an ESA effort and following an open-source approach in the frame of the SPINE community, SPIS knows today a real and dynamic community life. SPINE counts more than 250 registered members today, inside and outside EU. The average number of posts on the forums overtakes 70 messages per month.

Originally designed to focus on scientific applications, the application scope of SPIS is largely wider now and is regularly extended to new engineering applications or domains of physics. This includes, for instance, the modelling of electrical propulsion systems, ESD prediction on solar arrays or link with radiation models through deep charging phenomena.

The prediction of the electrostatic charge (absolute and relative) of spacecraft for engineering purposes is also a key issue with modern platforms that are more and more complex, operating high-power and sensitive electronic devices or using modern materials. Differential charging can lead to arcing, dangerous for the electronic payload. Absolute charging can induce disturbances on the radio transmission and/or the positioning systems using electric propulsion. This need is especially critical for GEO and MEO missions, where are located most of the commercial platforms. Moreover, the progressive generalisation of electrical propulsion systems on commercial platforms pushes integrators to perform much more detailed electrostatic analysis before the flight.

Legacy tools currently used in the industry, like NASCAP, cannot address these modern constraints. Thanks to its modularity and the implemented models representing the present state-of-the-art in plasma-surface interactions, SPIS is currently probably the best basis to address these issues in a self-consistent manner. There is a real need of a specific version of SPIS, called SPIS-GEO, to model MEO and GEO missions that would simplify its usage in an engineering context.

2.2. System workflow: the modelling chain

In order to perform a simulation with SPIS, the user has to follow a given number of pre- and post-processing steps in the correct order. In SPIS-CORE, the number of the steps to perform the simulation is high but provides the possibility to configure the simulation with the maximum of details, which is the objective of a simulation software targeted to scientists.

In SPIS-GEO, this modelling chain has to be simplified so that engineers can perform the simulation without requiring advanced knowledge in plasma physics or the numerical models. The following steps (illustrated in Figure 1) are proposed to define the simplified modelling chain of SPIS-GEO:

- **Geometry edition/creation:** during this step, the user either provides an existing CAD file or creates a new one from the definition of the satellite geometry
- **Mesh generation:** the user specifies the global refining factor to be applied to the meshing and if he/she wants the generated mesh to be analysed. Then he/she executes the mesh generation.
- **Mesh inspection:** the quality of the previously generated mesh is analysed if required. If the mesh quality is not satisfactory, the user can either modify the global refining factor or the spacecraft geometry definition.
- **Local properties and electrical circuit settings:** the user sets the local properties to be applied (material properties, boundary conditions, electrical nodes, electric propulsion, etc.) and defines the relationship between the different electrical nodes.

- **Environment and numerical settings:** the user has to provide the environment conditions in which he/she wants to model the satellite/plasma interaction as well as numerical settings (such as simulation duration).
- **Simulation loop:** the simulation loop is the phase during which the numerical core of SPIS-GEO is executed to compute the spacecraft/plasma interactions
- **Monitoring:** during the simulation loop, the user can visualize the progress of the computation as well as some key results (such as spacecraft potential as a function of time, collected currents as a function of time, etc.)
- **Post-processing:** once the simulation has complete, the user can visualize the results and export them in various formats.

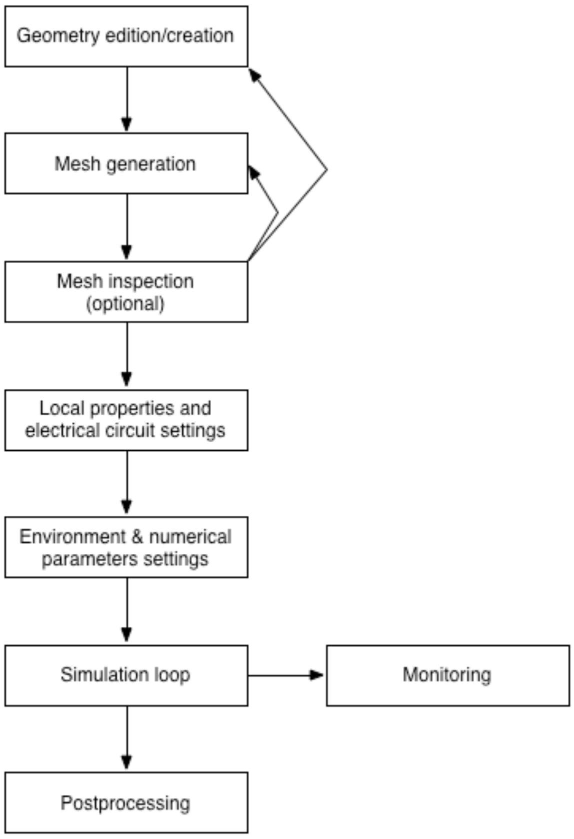
Geometry edition/creation
Mesh generation
Mesh inspection (optional)
Local properties and electrical circuit settings
Environment & numerical parameters settings
Simulation loop
Monitoring
Postprocessing

Figure 1: SPIS-GEO modelling chain

2.3. System data flow

This modelling chain can be described in more details by providing the flow of data between the different steps of the workflow, as illustrated in Figure 2. The different data that SPIS-GEO deals with are:

- **External input data:**
    - Spacecraft geometry description and/or an existing CAD file: to start a simulation in SPIS, one needs to model the spacecraft geometry, which should either be created from spacecraft geometry specification or from an existing CAD file.

  - Spacecraft environment: the definition of the spacecraft environment that can be deduce from its orbit.
  - Spacecraft properties: the material properties of the external spacecraft surfaces and the electric propulsion characteristics, if any.

- **Internal data:**
  - Geometry description: the geometry description of the spacecraft converted in GMSH-compatible format.
  - Unstructured 3D mesh: the 3D tetrahedral mesh generated by GMSH mesher, with the CAD groups information.
  - Mesh inspection results: the quality analysis results performed on the mesh.
  - Mesh with local properties applied: the 3D mesh and the various local properties (material properties, boundary conditions, etc.) defined by the user.
  - Environment and numerical parameters: the environment characteristics and the simulation parameters (e.g.: simulation duration)
  - Monitoring data: the relevant data to monitor the simulation and diagnose the convergence of the algorithms.
  - Simulation results: the physical results of the simulation, including potential in volume and on surface elements, the collected currents, the various particle population densities, etc.

- **External output:**
  - Exported simulation data: the simulation results exported in files for post-processing with external tools (e.g. VTK or ASCII files)

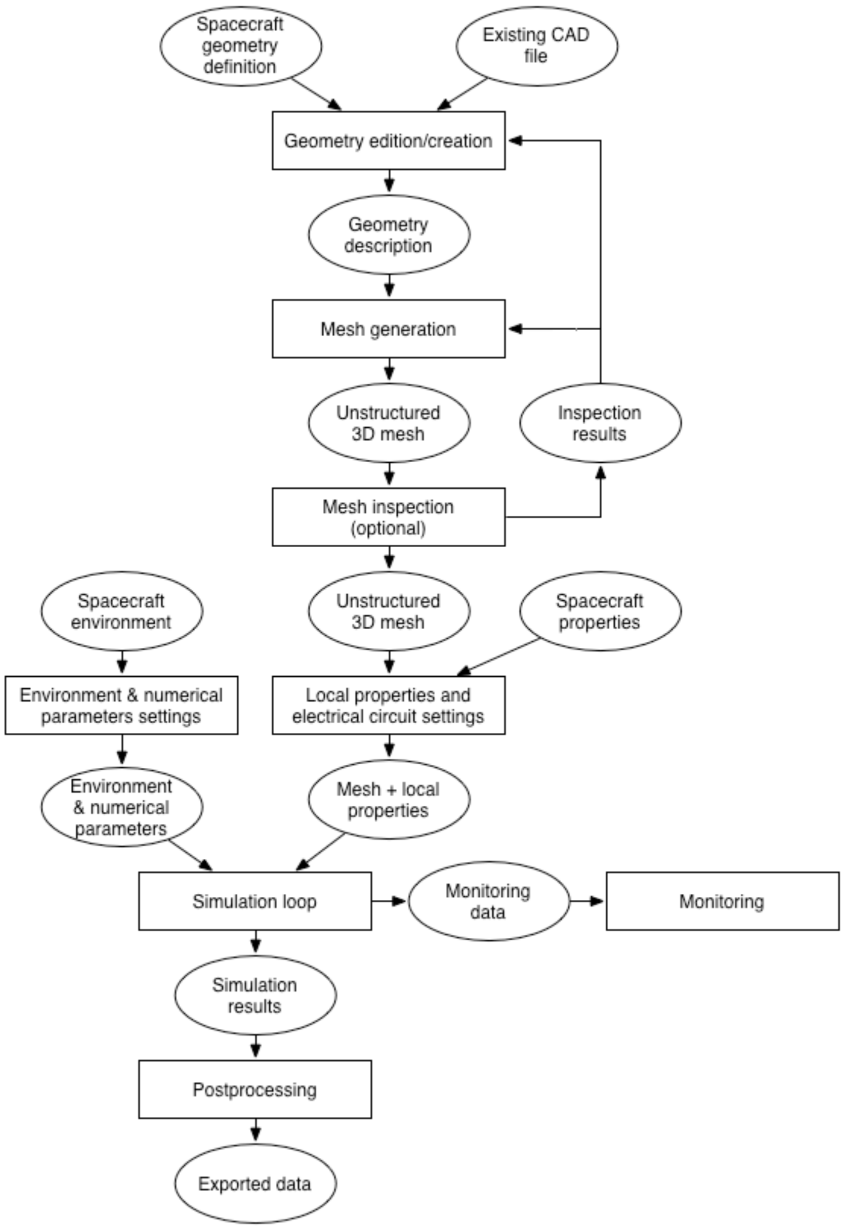

Figure 2: SPIS-GEO data flow

3. System design

3.1. Modular architecture

Modular programming is a software design technique that increases the extent to which software is composed of separate, interchangeable components, also called modules. Conceptually, modules represent a separation of concerns, and improve maintainability by enforcing logical boundaries between components. Modules are typically incorporated into the program through interfaces. A module interface expresses the elements that are provided and required by the module. The elements defined in the interface are detectable by other modules. The implementation contains the working code that corresponds to the elements declared in the interface.

The OSGi framework [http://www.osgi.org] is a module system and service platform for the Java programming language that implements a complete and dynamic component model. Its first specification was released in 2000. In 2003 Eclipse [http://www.eclipse.org] selected OSGi as the underlying runtime for their plug-in architecture. A very dynamic open-source community and years of experience in multiple enterprise applications (e.g. Glassfish [https://glassfish.dev.java.net/]) make the framework now mature.

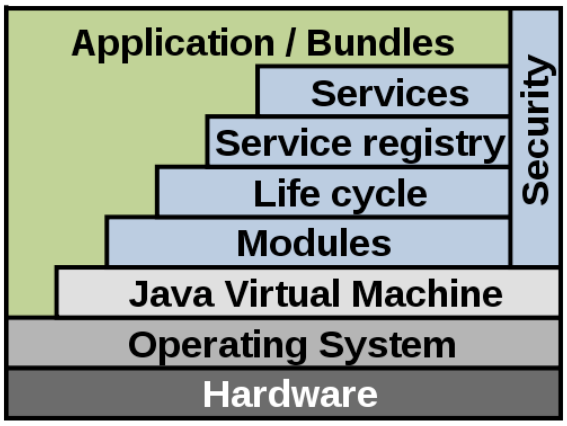

Figure 3: OSGi Services Platform architecture

The OSGi Platform can be divided in two main elements:

- A services platform
- A deployment infrastructure

A services platform is defined as a software platform that supports the service orientation interaction depicted in Figure 4. This interaction involves three main actors: service providers, service requesters and a service registry, although only the service registry belongs to the services platform. In the service orientation interaction, service providers publish service descriptions, and service requesters discover services and bind to the service providers. Publication and discovery are based on a service description. In OSGi, service providers and requesters are part of the bundle that is both a logical as well as physical entity.

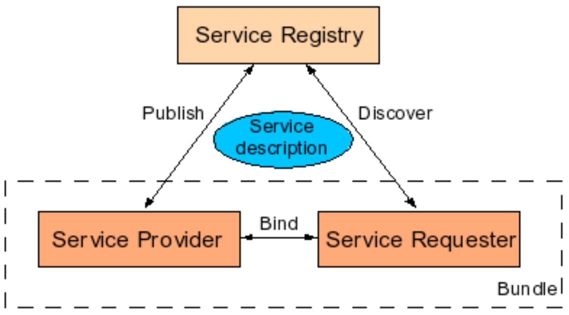

Figure 4: Services oriented interactions

Physically, a bundle corresponds to a delivery and deployment unit that is materialized by a JAR file that contains code and resources (i.e., images, libraries, etc.) along with a file that contains information about the bundle, the manifest file. The OSGi framework provides mechanisms to support continuous deployment activities. These deployment activities include installation, removal, update, starting (activation) and stopping (de-activation) of a physical bundle. Once a bundle is installed in the platform, it can be activated if deployment dependencies that are associated to the bundle are fulfilled. The lifecycle of a bundle is described in Figure 5.

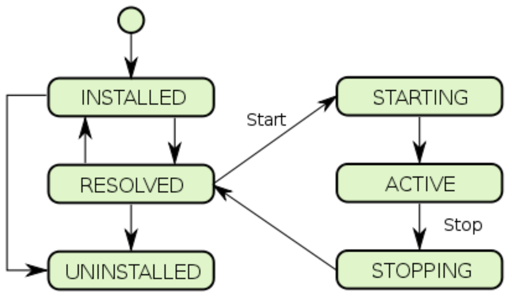

Figure 5: OSGi bundle lifecycle

There are numerous advantages of using modular programming:

- Higher granularity encapsulation: the modules add a layer of abstraction that makes it easier to substitute components and enables a better reuse and sharing of code. As the “contract” of a module (what it provides and what it requires) is clearly defined, they are easier to integrate in other contexts.
- Simplified distributed team development: the bundle isolation makes it much easier for their development by different teams. The huge number of the Eclipse plug-ins, developed by numerous companies illustrates the extent of the distributed development possibilities.
- Lower maintenance cost: bundles can be updated individually and as their boundaries are clearly expressed and enforced by the OSGi framework, updating one component has limited impact on others.

Other advantages provided by OSGi are:

- The possibility, in future versions of SPIS, to deploy the bundles in distributed environments
- The easier integration with existing tools in TEC-EES section (ESABase 2)

3.2. Keridwen 2.0

Keridwen 2.0 [http://www.keridwen.org] is the new generation of open-source Integrated Modelling Environment (IME). Based on OSGi, it implements core functionalities for scientific applications like SPIS-GEO. As illustrated on Figure 6, an application based on Keridwen is formed of OSGi components with a dedicated user interface and controller. They are plugged dynamically to Keridwen central controller and graphical user interface and can rely on Keridwen common services and data manager to execute non tailored activities, favouring reuse and validation of core components.

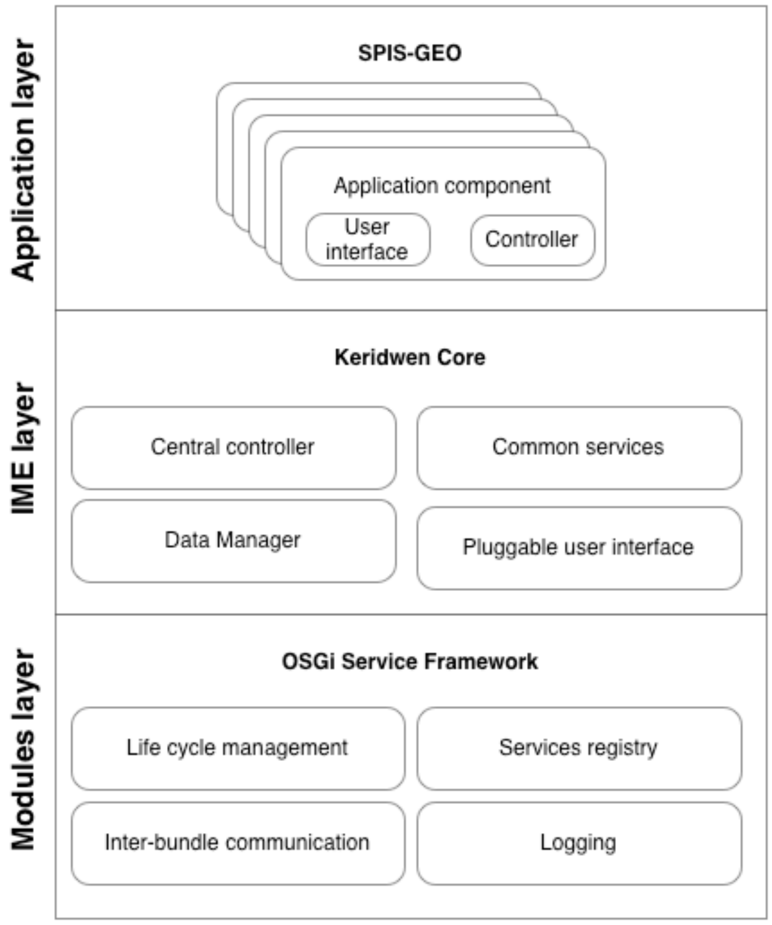

Figure 6: Keridwen-based architecture

It is proposed to base the architecture of SPIS-GEO on this new version of Keridwen. The objective is to benefit from this component-based and service-oriented architecture to facilitate SPIS future development and interactions with other tools.

In order to design the architecture of SPIS-GEO, the software has to be decomposed in components, which is the subject of the next section. The components identified are very close to existing SPIS Tasks. This will allow reusing most of the existing and validated code in SPIS-GEO, only changing the packaging of the modules from the Tasks formalism to the OSGi bundle standard.

4. Software components description

As described in Figure 1 and Figure 2, SPIS-GEO workflow is composed of unitary steps that depend on each other according to a modelling chain. It is thus very natural, as a first step, to map SPIS-GEO components to these steps. In the future, components could be split further if required.

Here is a description of each of the software components.

4.1. Geometry module

Name	Geometry module
Input	• Spacecraft geometry definition • And/or an existing CAD file
Functions	• Import the existing CAD file and convert it into GMSH-compatible geometry • Provide user assistance to create and modify GMSH geometries (simple sphere, cube, cylinder, thin panels and booms, etc.) • Preview the geometry and its groups
Dependencies	• GMSH for geometry design • VTK for group visualization
Output	• GMSH-compatible geometry description

4.2. Mesh generation module

Name	Mesh generation module
Input	• GMSH-compatible geometry description
Functions	• Setup of the global refining factor • Mesh generation
Dependencies	• GMSH for the meshing • JFreeMesh for the mesh conversion
Output	• Unstructured 3D mesh

4.3. Mesh inspection module

Name	Mesh inspection module
Input	• Unstructured 3D mesh
Functions	• Analyse mesh quality • Provides graphical illustration of quality criteria
Dependencies	• JFreeMesh for the mesh analysis • VTK for the mesh visualization
Output	• Unstructured 3D mesh

4.4. Local properties module

Name	Local properties module
Input	• Unstructured 3D mesh
Functions	• Local properties database management (creating, editing and deleting material properties) • Local properties allocation to spacecraft groups • Groups visualization • Electric propulsion settings • Spacecraft circuit definition
Dependencies	• VTK for group visualization • JFreeMesh for local properties allocation on mesh elements
Output	• Unstructured 3D mesh with local properties applied

4.5. Environment and numerical parameters module

Name	Environment and numerical parameters module
Input	• Spacecraft environment
Functions	• Definition of the plasma environment • Selection of worst-case or typical MEO/GEO environments • Solver models and numerical parameters definition • Spacecraft illumination • Activation of transition scenarios
Dependencies	N/A
Output	• Global environment and numerical parameters

4.6. Numerical kernel module (SPIS-NUM)

Name	Numerical kernel module (SPIS-NUM)
Input	• Unstructured 3D mesh with local properties applied • Global environment and numerical parameters
Functions	• Compute spacecraft-plasma interactions of the provided system
Dependencies	N/A
Output	• Simulation results • Monitoring data

4.7. Monitoring module

Name	Monitoring module
Input	• Monitoring data

Functions	• Real-time monitoring of the running simulation
Dependencies	• JFreeChart for 2D diagram display
Output	N/A

4.8. Post-processing module

Name	Geometry module
Input	• Simulation results
Functions	• Visualization of the simulation results • Generation of the simulation report • Export of the data in various formats (VTK, ASCII, images)
Dependencies	• Keridwen reporting for reports generation • VTK for data visualization • JFreeChart for 2D diagram visualization
Output	• Exported data (VTK, ASCII, images, reports)

5. Third-party components

5.1. GMSH

Gmsh [http://geuz.org/gmsh/] is an open-source 3D finite elements grid generator with a build-in CAD engine. It is currently used by SPIS for the CAD modelling and the mesh generation.

5.2. VTK

The Visualization Toolkit [http://www.vtk.org/] is an open-source, software system for 3D computer graphics, image processing and visualization.

It is used by SPIS in Cassandra [http://dev.artenum.com/projects/cassandra/], an open source scientific data viewer to visualize 3D data (spacecraft groups and simulation results for instance).

5.3. JFreeChart

JFreeChart [http://www.jfree.org/jfreechart/] is an open source Java chart library that makes it easy for developers to display professional quality charts in their applications. It is used in SPIS to display 2D charts.

5.4. Keridwen data manager

Keridwen data manager is a component that offers data access capabilities to scientific applications. It is designed to allow concurrent and remote access to any kind of data, as well as data persistency.

5.5. Keridwen command line

Kerdiwen command line is a command line utility based on JRosetta [http://dev.artenum.com/projects/JRosetta] that allows controlling Keridwen modules from scripts.

5.6. Keridwen reporting

Keridwen reporting is a reports generation module that builds pre-formated PDF files from simulation results.

5.7. Keridwen wizards

Keridwen wizards module, based on Shaman [to be released soon], is a wizard engine that automatically controls wizard panels sequence from a XML configuration file.

5.8. JFreeMesh

JFreeMesh [http://dev.artenum.com/projects/JFreeMesh] is a 3D mesh library written in Java and providing a high level API for mesh manipulation. It is used in SPIS for instance for mesh inspection and local properties allocation.

6. Software requirements versus components compliance matrix

Component	Addresses SR #
Geometry module	SR-SC-003, SR-SC-005
Mesh generation module	SR-SC-004
Mesh inspection module	SR-GE-004
Local properties module	SR-SC-001, SR-SC-002, SR-SC-006, SR-SC-007, SR-SC-008
Environment and numerical parameters module	SR-EN-001, SR-EN-002; SR-EN-003, SR-EN-004, SR-EN-005
Numerical kernel module (SPIS-NUM)	SR-EN-001, SR-EN-002; SR-EN-003, SR-EN-004, SR-EN-005, SR-SM-001, SR-SM-002, SR-RP-001
Monitoring module	SR-SM-002
Post-processing module	SR-PP-001
Keridwen core components	SR-EX-002; SR-EX-003; SR-EX-004
GMSH	SR-SC-003, SR-SC-005
VTK	SR-SC-002, SR-PP-001
JFreeChart	SR-SM-002, SR-PP-001
Keridwen data manager	SR-PR-001, SR-PR-002, SR-PR-003, SR-PR-004
Keridwen command line	SR-EX-001
Keridwen reporting	SR-PP-001
Keridwen wizards	SR-GE-003, SR-GE-004, SR-GE-005
JFreeMesh	SR-SC-001, SR-SC-002

7. Conclusion

SPIS-GEO application is proposed to be built on top of Keridwen 2.0, an OSGi-based Integrated Modelling Environment to achieve a modular application, easier to maintain and to integrate with other tools.

A description of the components of this modular application has been given as well as the compliance matrix with the software requirements.