*Open Simulation Framework*

# *openSF*

# INTERFACE CONTROL DOCUMENT

| | |
|---|---|
| **Code :** | openSF-DMS-ICD-001 |
| **Issue :** | 3.0.1 |
| **Date :** | 14-05-2018 |

| | Name | Function | Signature |
|---|---|---|---|
| **Prepared by** | Rui Mestre | Project Engineer | |
| **Reviewed by** | Federico Letterio | Project Manager | |
| **Approved by** | Federico Letterio | Project Manager | |
| **Signatures and approvals on original** | | | |

DEIMOS Space S.L.U.
Ronda de Poniente, 19, Edificio Fiteni VI, 2-2ª
28760 Tres Cantos (Madrid), SPAIN
Tel.: +34 91 806 34 50 / Fax: +34 91 806 34 51
E-mail: deimos@deimos-space.com

This page intentionally left blank

# Document Information

| Contract Data | |
|---|---|
| **Contract Number:** | 22852/09/NL/FF |
| **Contract Issuer:** | ESA/ESTEC |

| Internal Distribution | | |
|---|---|---|
| Name | Unit | Copies |
| Jose Antonio González Abeytua | Head of the Earth Observation Systems Business Unit | 1 |
| Ricardo Moyano | Earth Observation Systems Business Development | 1 |
| Enrique del Pozo | Earth Observation Systems | 1 |
| Rui Mestre | Earth Observation Systems | 1 |
| Internal Confidentiality Level (DMS-COV-POL05) | | |
| Unclassified ☑ | Restricted ☐ | Confidential ☐ |

| External Distribution | | |
|---|---|---|
| Name | Organisation | Copies |
| Raffaella Franco | ESA/ESTEC | 1 |
| Michele Zundo | ESA/ESTEC | 1 |
| Montserrat Piñol | ESA/ESTEC | 1 |
| Maurizio de Bartolomei | ESA/ESTEC | 1 |

| Archiving | |
|---|---|
| Word Processor: | MS Word 2000 |
| File Name: | openSF-DMS-ICD-001-30.doc |

# Document Status Log

| Issue | Change description | Date | Approved |
|---|---|---|---|
| 1.0 | First version of this document | 28/10/2009 | |
| 1.0 | Updated from ESA comments:<br>❑ Removed RESTRICTED confidentiality level from the document, the new level is UNCLASSIFIED.<br>❑ Added a XSD schema file for XML configuration files validation (section 4.7).<br>❑ Updated figure: Integration of models and tools in openSF platform<br>❑ Added new section OpenSF System Description that covers the framework overview, application architecture and OSFI description.<br>❑ Third party tool interface moved to Advanced Interfaces (section 4.10)<br>❑ Updated section 4.7, double range values changed<br>❑ Added section 4.8 Example of use. | 12/11/2009 | |
| 1.1 | Updated from ESA comments:<br>❑ Help system documentation updated. Section 4.9.1.1<br>❑ New section added for Other Programming languages support. OpenSF user hints added for Matlab and IDL. (section 3.2.1) | 23/11/2009 | |
| 1.2 | Updated from openSF AR 2:<br>❑ Updated section 3.2.1 reflecting new supported languages.<br>❑ Added new section explaining the adoption of Earth Explorer File Format. | 12/11/2010 | |
| 2.0 | Updated for openSF CCN1 PDR:<br>❑ Added section 4.9.3 reflecting XML format for defining openSF element externally;<br>❑ Added section 3.3 reflecting OSFEG libraries configuration file format. | 18/04/2013 | |
| 2.1 | Updated from ESA comments to CCN1 PDR:<br>❑ Updated section 1.1 stating of the added interfaces for openSF v3. | 05/06/2013 | |
| 2.2 | New version including the integration of Python modules in openSF. | 04/04/2014 | |
| 3.0 | Updated to make the document compliant with ESA generic E2E simulator ICD. | 26/05/2015 | |
| 3.0.1 | (editorial) Removed reference to ndims on page 41<br>Corrected obsolete version of EO FFSTD page 11 | 14/05/2018 | |

# Table of Contents

## List of Tables

## List of Figures

# 1. INTRODUCTION

## 1.1. Purpose

The objective of this document is to present and describe the integration requirements of models into the open Simulation Framework.

It details the interfaces provided and needed, allowing scientific models and product exploitation tools to be plugged in the system platform with ease as well as some communication links with other software components.

This document has been produced in the frame of the "open Simulation Framework", project developed by DEIMOS Space S.L.U. that inherits from another ESA project, the end-to-end mission/system performance Simulation for EarthCARE (ECSIM - ESA's contract number 20003/065/NL).

This document was updated for openSF V3.3 by making it compliant with ESA generic E2E simulator ICD [AD-E2E].

## 1.2. Scope

The applicability of this document begins once openSF is suitable to be adapted for a specific purpose simulator or processing chain.

This document is divided in the following sections:

- Section 1 (this one) provides a glimpse on the document contents and purposes.
- Section 2 establishes the relations of this document with other documents and standards.
- Section 3 gives an overview of the openSF system and its relationships with other projects.
- Section 4 details the openSF interfaces and gives some development guidelines.

Potential readers of this document include scientists/engineers and modelers interested in using the openSF added-value capabilities with their developments.

## 1.3. Acronyms and Abbreviations

The acronyms and abbreviations used in this document are the following ones:

*Table 1-1: Acronyms and abbreviations*

| Acronym | Description |
|---|---|
| AD | Architectural Design<br>Applicable Document |
| ADD | Architectural Design Document |
| API | Application Programming Interface |
| AD | Applicable Document |
| COTS | Commercial Off-The-Shelf |
| CPU | Central Processing Unit |
| DBMS | Database Management System |
| DMS | DEIMOS Space |
| E2E | End-to-End Simulator |
| ECSIM | EarthCare end-to-end SIMulator |
| ESA | European Space Agency |
| GUI | Graphical User Interface |
| HMI | Human machine Interface |
| HW | Hardware |
| I/F | Interface |
| I/O | Input/Output |
| ICD | Interface Control Document |
| MoM | Minutes of Meeting |
| OSFI | Open Simulation Framework Integration Libraries |
| PDR | Preliminary Design Review |
| RD | Reference Document |
| SIM | System Installation Manual |
| SUM | System User Manual |
| SOW | Statement Of Work |
| SR | Software Requirements |
| SRD | Software Requirements Document |
| TBC | To Be Confirmed |
| TBD | To Be Defined / Decided |

## 1.4. Definitions

The definitions of the specific terms used in this document are the following ones:

***Table 1-2: openSF relevant definitions table***

| Definition | Meaning |
|---|---|
| **Batch mode** | It is the capability of the simulator to perform consecutive runs without continuous interactions with the user. Batch mode checks the agreement or not between the output of a given module and the input by the next one in the sequence of the simulation. Several modes of executions can be performed: ❑ Iteratively, executing one or more simulations ❑ Iteratively, executing the same simulation several times depending on the parameters configuration ❑ Same as above but by executing a batch script. |
| **Configuration File** | A small XML file that contains all the parameters necessary to execute a model. A configuration file instance must comply with the corresponding XML schema defined at model creation time. |
| **Framework** | Software infrastructures designed to support and control the simulation definition and execution. It includes the GUI, domain and database capabilities that enable to perform all the functionality of the simulator. |
| **Model** | Executable entity that can take part in a simulation. A model can be understood, broadly speaking, also as an "algorithm". Basically, it contains the recipe to produce products function of inputs. A model contains also several rules to define the input, output and associated formats. Furthermore, its behaviour is controlled by one configuration file. Overall, the architecture of a model consists of: ❑ The source code and its binary compiled counterpart ❑ A configuration file with its parameters ❑ An input file that characterizes its inputs ❑ An output file that characterizes its outputs Models are not considered part of the framework. |
| **OSFI** | To integrate an external model into openSF, the models need to fulfill a series of interface requirements. The Open Simulation Framework Integration Libraries (OSFI from now on) will be used to ease the integration of models into the framework. The Integration Libraries activity will provide the model developer with a set of routines with a well-defined public interface hiding the implementation details. This set of routines is currently available in C++, ANSI C and Fortran 90. |
| **Parameter** | A constant whose value characterizes a given particularity of a model. Parameters are user-configurable, they are fixed before launching a model and, for practical reasons, and not all of them shall be accessible from the HMI. |

| Definition | Meaning |
|---|---|
| **Session** | A session is defined as an execution of a simulation, an ordered set of simulations or an iterative execution of simulation(s) with different parameter values. There are no restrictions on how to concatenate these simulations, they do not have to be compatible between them but, if necessary, the final output files of a simulation can be used by the following simulation. |
| **Simulation** | A simulation is understood as a list of models (or even a model alone) that is run sequentially and produces observable results. |
| **Tool** | A tool is an external executable file that performs a given action to a certain group of files. Used into the openSF platform and associated to a certain file extension these tools can be called to perform off-line operations to products involved in simulations. |

# 2. RELATED DOCUMENTS

## 2.1. Applicable Documents

The following table specifies the applicable documents that were compiled during the project development.

*Table 2-1: Applicable documents*

| Reference | Code | Title |
|---|---|---|
| [AD-E2E] | PE-ID-ESA-GS-464 | ESA generic E2E simulator Interface Control Document |
| [AD-SRD] | openSF-DMS-SRD-001 | OpenSF System Requirements Document |
| [AD-SUM] | openSF-DMS-SUM-001 | OpenSF System User Manual |
| [AD-ADD] | openSF-DMS-ADD-001 | OpenSF Architecture Design Document |

## 2.2. Reference Documents

The following table specifies the reference documents that shall be taken into account during the project development.

*Table 2-2: Reference documents*

| Reference | Code | Title | Issue |
|---|---|---|---|
| [RD-OSFI-DM] | OSFI-DMS-TEC-DM | OpenSF Integration Libraries Developers Manual | 1.7 |
| [RD-OSFEG-DM] | OSFEG-DMS-TEC-DM | OpenSF Error Generation Libraries Developers Manual | 1.1 |
| [RD-MATLAB] | MATLAB-API-EXTERNAL-IF | Matlab External Interfaces | Version 6 |

## 2.3. Standards

The following table specifies the standards that were complied with during project development.

*Table 2-3: Standards*

| Reference | Code | Title | Issue |
|---|---|---|---|
| [E40C] | ECSS-E-40C | Software development Standard | 3 - 6 March 2009 |
| [BNF] | (see also en.wikipedia.org /wiki /Backus-Naur_form) | Algol-60 Reference Manual | 5, 1979 |
| [XML] | (www.w3.org/TR/xml11/) | Extensible Markup Language (XML) 1.1 | Second Edition , Sep 29 2006 |
| [XSD] | (http://www.w3.org/TR/xmlschema-2/) | XML Schema Definition Language | Oct 28 2004 |
| [EO-FFS] | PE-TN-ESA-GS-0001 | Earth Observation File Format Standard | 3.0 |

# 3. OPENSF SYSTEM DESCRIPTION

This section gives an overview of openSF structure, a brief description of implementation and design methods, an introduction to the openSF Integration Libraries (OSFI) and to the openSF Error Generation Libraries (OSFEG).

## 3.1. System Overview

In the frame of concept and feasibility studies for the Earth Observation (EO) activities, mission performance in terms of final data products needs to be predicted by means of so-called end-to-end (E2E) simulators.

A specific mission E2E simulator is able to reproduce all significant processes and steps that impact the mission performance and obtains as output simulated final data products.

OpenSF is a generic simulation framework product being developed by DEIMOS aimed to cope with these major goals. It provides end-to-end simulation capabilities that allow assessment of the science and engineering goals with respect to the mission requirements.

Scientific models and product exploitation tools can be plugged in the system platform easily using a well-defined integration process. Figure 3-1 sketches this simple mechanism.

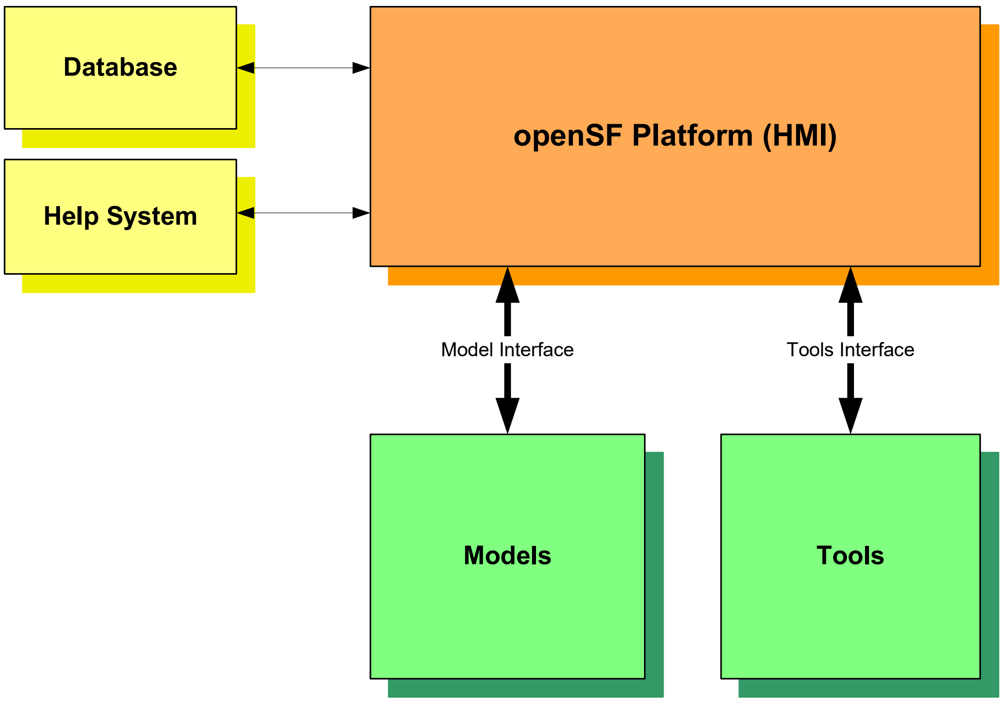


*Figure 3-1: Integration of models and tools in openSF platform*

Users can integrate every kind of executable as either a model or product tool (as defined in section 4.1 and 4.3 respectively) because openSF does not apply any programming language dependency. The integration is made at Operating System level therefore it implies a minimal intrusion of the models' software code.

## 3.2. OSFI Libraries

For helping model developers, openSF provides a set of libraries to ease the integration of models into the framework. These libraries are known as **OSFI** (**OpenSF Integration libraries**) and are a set of utilities programmed in different languages that handle the interfacing problems between models and openSF.

The Integration Libraries provide the models' developers with a set of routines with a well-defined public interface hiding the implementation details. This set of routines has been developed in C++, but there are currently available wrappers for C, Fortran 90 and Fortran 77. The mechanism used for integrating IDL, Matlab and Python models is described in [AD-ADD] and [RD-OSFI-DM]. A developer's manual is also available detailing the use of the libraries [RD-OSFI-DM].

The programming tasks covered by OSFI libraries are:

- XML Configuration files parsing (based on XERCES-C libraries)
- Command line parsing following openSF convention.
- Log message and error handling mechanism.

Some of the OSFI libraries goals are:

- Enhance the model developer efficiency and productivity providing an XML configuration files parser, a command line parser and an easy log messages interface.
- Provide native support for models implemented in C/C++, Fortran 90, Fortran 77, IDL, Matlab and Python.
- Ensure the model compatibility with openSF avoiding problems derived from coding errors.
- Ensure that no memory sharing mechanism is used.
- Availability of binary packages for multiple target platforms (UNIX, OSX).

Figure 3-2 outlines how OSFI libraries solve the problem of openSF integration.

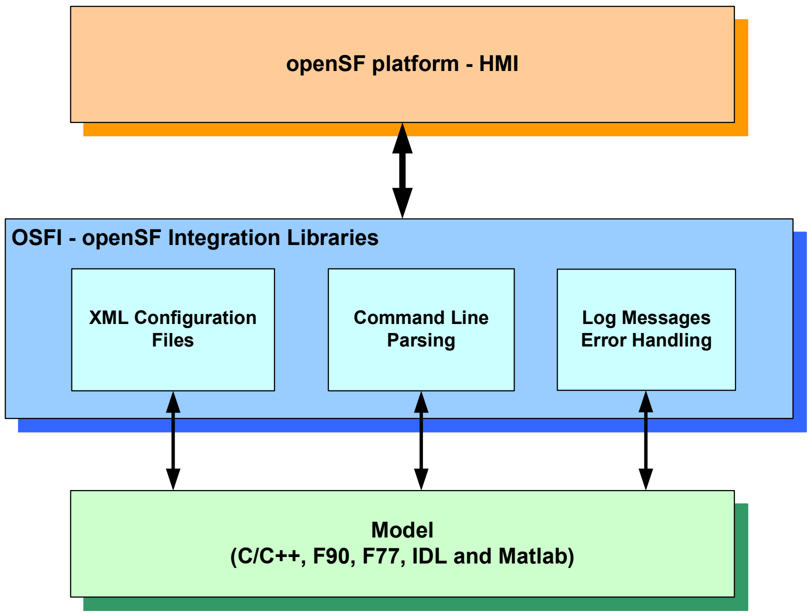

openSF platform - HMI
OSFI - openSF Integration Libraries
XML Configuration Files
Command Line Parsing
Log Messages Error Handling
Model
(C/C++, F90, F77, IDL and Matlab)

*Figure 3-2 OpenSF - OSFI Integration*

Software developers shall choose whether OSFI libraries are suitable for the model implementation or handle the integration issue by themselves. The model development process is outlined in Figure 3-3.

The models definition and the simulation chain definition/execution in openSF HMI are described in the [AD-SUM].

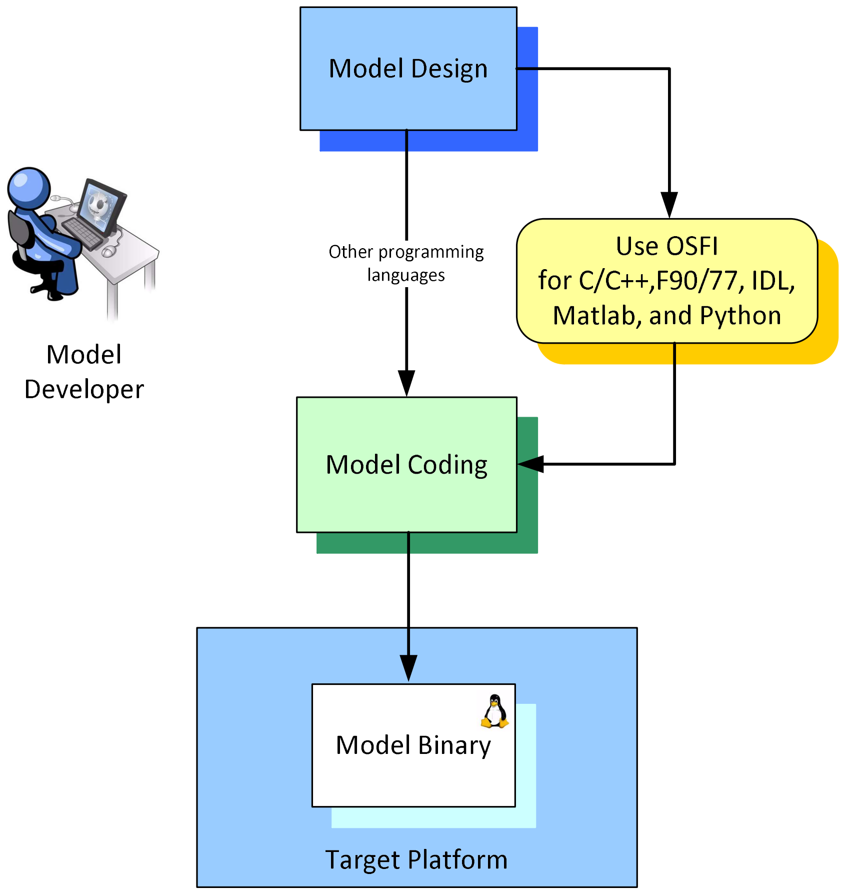

Model Design
Model Developer
Other programming languages
Use OSFI for C/C++,F90/77, IDL, Matlab, and Python
Model Coding
Model Binary
Target Platform

***Figure 3-3: Model Development Process***

## 3.2.1. Other Programming Languages

As mentioned before the integration libraries are currently available in five programming languages (C/C++, Fortran 90, Fortran 77, IDL, Matlab, and Python). As openSF does not have any programming language dependency it is possible to integrate within it models developed in other languages such as Java, Mathematica and others, as long as these models follow the openSF interface convention described in section 4.

In order to ease the integration of models programmed in other languages some implementation guidelines are detailed below.

### 3.2.1.1. Java, Mathematica and others

The steps that shall be followed for proper openSF integration are:

1. Create an XML parser for openSF configuration files (create a new parser or wrapper the existing one in OSFI libraries[1])
2. Implement a function that formats the model log/error messages following the openSF log convention.
3. Compile the model as an executable file taking into account the command line invocation format described in [AD-E2E].

## 3.3. OSFEG libraries

The Open Simulation Framework Error Generation Libraries (OSFEG from now on) can be used as a tool to ease the mathematical modeling of a perturbation within statistical analysis scenarios. OSFEG offers to developers a well-documented interface to ease the modeling and generation of a perturbation over desired parameters. The libraries provide an error-modeling interface based on a XML file definition and its correspondent implementation in C++. A developer's manual is available detailing the use of the libraries [RD-OSFEG-DM].

This section describes the mathematical functions implemented within the error generation libraries. The libraries include the most used analytical and random functions for parameter perturbation in E2E simulation modeling scenarios. The parameter perturbation functions are defined through an XML file. An example is shown at the end of this section. This section is especially relevant because the error definition file describes the mathematical behavior of the parameters perturbation. It is also included a detailed description of the variables involved in the function definition.

### 3.3.1. *Error Functions*

#### 3.3.1.1. Deterministic Functions

Deterministic functions are those whose value is known in the entire time domain.

- **Affine**

Calculate the perturbation as an affine value. An affine transformation consists in a linear transformation and a translation.

- $error = a_1 + a_0 * t$

```
<affine>
  <float value="1" />  <!--  Linear Transformation Variable a0 -- >
  <float value="1" />    <!--  Translation Variable  a1 -- >
</affine>
```

[1] Most of the programming languages have interfaces for linking with C/C++ libraries.

❑ **Bias**

Calculate the perturbation as a constant value.

```
<bias>
  <float value="1" />  <!-- Constant Value -- >
</bias>
```

❑ **Linear**

Calculate the perturbation as a linear value:

- $p = a * t$

This is a particular case of affine transformation when translation variable is equals to 0.

```
<linear>
  <float value="1" />  <!-- Linear Transformation Variable a -- >
</linear>
```

❑ **Parabolic**

Calculate the perturbation as a parabolic value.

- $p = a_0 + a_1 * t + a_2 * t^2$

```
<parabolic>
  <float value="1" />   <!-- a0 -->
  <float value="1" />   <!-- a1 -->
  <float value="1" />   <!-- a2 -->
</parabolic>
```

❑ **Polynomial**

Calculate the perturbation as a generic polynomial value. This function has as many float parameters as degrees of the desired polynomial plus one.

```
<polynomial>
  <float value="1" />   <!-- a0 -->
  <float value="1" />   <!-- a1 -->
     …
  <float value="1" />   <!-- a(n-2) -->
```

```
   <float value="1" />   <!-- a(n-1) -->
</polynomial>
```

- **Step**

Calculate the perturbation as step function.

- if simTime < t => $p = a_0$
- if simTime > t => $p = a_1$

```
<step>
   <float value="3" />    <!-- t -->
   <float value="1" />    <!-- a0 -->
   <float value="-1" />   <!-- a1 -->
</step>
```

- **Sinusoidal**

Calculate the perturbation as sinusoidal function

- p = a * sin(2 * pi * f * t + phi)
- f(Hz)
- phi(deg)
- t(secs)

```
<sinusoidal>
   <float value="10" />   <!-- Amplitude a -->
   <float value="10" />   <!-- Frequency f in Hz -->
   <float value="0" />     <!-- Angle phi in deg. -->
</sinusoidal>
```

- **Tangent**

Calculates the perturbation as tangent function

- p = a * tan(2 * pi * f * t + phi)
- f(Hz)
- phi(deg)
- t(secs)

Remember that the tangent function have singularities when the angle evaluated is (+-)*n*pi/2.

```
<tangent>
   <float value="10" />   <!-- Amplitude a -->
   <float value="1" />     <!-- Frequency f in Hz -->
   <float value="0" />     <!-- Angle phi in deg. -->
</tangent>
```

### 3.3.1.2. Sampling Functions

Error Generation libraries implements three interpolation methods, linear, polynomial and spline sampling.

In order to define the points of the interpolation there is a common set of variables that are listed below.

- xMin: Min value of abscise axis
- xMax: Max value of abscise axis
- step: Increment between abscise values

The number of points must be:

$$\frac{xMax - xMin}{step} = nValues$$

- **Linear Sampling**

This function makes an interpolation with the given points assuming it follows a linear rule. In out of range values

```
<linearSampling xMin="1.0" xMax="10.0" step="1">
   <float value="1" />
   <float value="3" />
   <float value="5" />
   <float value="7" />
   <float value="3" />
   <float value="2" />
   <float value="2" />
   <float value="10" />
   <float value="4" />
   <float value="3" />
</linearSampling>
```

❑ **Polynomial Sampling**

This interpolation method builds a polynomial grade n, being n the number of specified points. This interpolation minimizes the Least Square Error. Ref: Neville Method.

```
<polynomialSampling xMin="1.0" xMax="10.0" step="1">
  <float value="1" />
  <float value="2" />
  <float value="1" />
  <float value="2" />
  <float value="1" />
  <float value="2" />
  <float value="1" />
  <float value="2" />
  <float value="1" />
  <float value="2" />
</polynomialSampling>
```

❑ **Spline Sampling**

Interpolate the given “n” points with Cubic Splines Method.

```
<splineSampling xMin="1.0" xMax="20.0" step="1">
     <float value="2" />
  <float value="3" />
  <float value="2" />
  <float value="3" />
  <float value="2" />
  <float value="3" />
  <float value="2" />
  <float value="3" />
  <float value="10" />
  <float value="2" />
  <float value="3" />
  <float value="2" />
  <float value="4" />
  <float value="7" />
  <float value="2" />
```

```
    <float value="3" />
    <float value="2" />
    <float value="3" />
    <float value="2" />
    <float value="7" />
</splineSampling>
```

#### 3.3.1.3. Nondeterministic Functions

These are common random function implementation with seed management for testing purposes.

- **Beta Distribution**

Generate random values with Beta function as probability density function.

```
<parameter name="Beta distribution">
    <beta seed="1" v="2" w="5" xMin="0.0" xMax="1.0" />
</parameter>
```

- **Gamma Distribution**

Generate random values with Gamma function as probability density function.

```
<parameter name="Gamma distribution">
    <gamma seed="1" location="0.0" scale="0.5" shape="9" />
</parameter>
```

- **Exponential Distribution**

Generate random values with Exponential function as probability density function.

```
<parameter name="Exponential distribution">
    <exponential seed="1" a="1" b="1.5" />
</parameter>
```

- **Normal Distribution**

Generate random values with Gaussian function as probability density function.

```
<parameter name="Normal distribution">
  <normal seed="1" mu="100.0" sigma="10.0" />
</parameter>
```

- **Uniform Distribution**

Generate random values following a Uniform Distribution.

```
<parameter name="Uniform distribution">
  <uniform seed="1" xMin="0" xMax="1" />
</parameter>
```

- **Poisson Distribution**

Return the perturbation as a generated random value with Poisson function as probability density function.

```
<parameter name="Poisson distribution">
  <poisson seed="1" mu="10" />
</parameter>
```

- **Truncated Gaussian Distribution**

Return the perturbation as a generated random value with Truncated Gaussian function as probability density function.

```
<parameter name="Truncated gaussian distribution">
  <truncatedGaussian seed="1" mu="0.5" sigma="0.2" xMin="0.4" xMax="0.6" />
</parameter>
```

- **Uniform Discrete Distribution**

Return the perturbation as a generated random value with Uniform Discrete function as probability density function.

```
<parameter name="Uniform discrete distribution">
```

```
  <uniformDiscrete seed="1" i="0" j="1" />
</parameter>
```

- **Distribution with custom Probability Density Function**

Returns the value of a random variable generated with a custom probability density function given. It is only recommended to use it by expert developers/scientists.

```
<parameter name="Custom PDF">
  <customPDF seed="24" xMin="0.0" xMax="12.0" step="1">
    <float value="7" />
    <float value="43" />
    <float value="21" />
    <float value="10" />
    <float value="2" />
    <float value="6" />
    <float value="23" />
    <float value="31" />
    <float value="7" />
    <float value="2" />
    <float value="7" />
    <float value="43" />
    <float value="21" />
  </customPDF>
</parameter>
```

#### 3.3.1.4. Binary and Composite Operations

Error Generation Libraries implements the basics mathematical operations in binary mode. The operations implemented are:

- **Addition**
- **Subtraction**
- **Multiplication**
- **Division**
- **Exponentiation**
- **Root**

Composite operations consist of a deterministic function with one or more of its parameters following another function or binary operation. An example of an error definition file implementing all the binary and some composite operations is shown below:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
  <parameter name="Affine and sinusoidal">
    <affine>
      <sinusoidal>
        <float value="10" />
        <float value="90" />
        <float value="0" />
      </sinusoidal>
      <float value="5" />
    </affine>
  </parameter>
  <parameter name="Sinusoidal and beta">
    <sinusoidal>
      <beta seed="1" v="1.0" w="2.0" xMin="10.0" xMax="15.0" />
      <float value="10" />
      <float value="0" />
    </sinusoidal>
  </parameter>
  <parameter name="Addition">
    <addition>
      <exponentiation>
        <sinusoidal>
          <float value="10" />
          <float value="90" />
          <float value="0" />
        </sinusoidal>
        <float value="2" />
      </exponentiation>
      <subtraction>
        <sinusoidal>
          <float value="40" />
          <float value="90" />
          <float value="5" />
        </sinusoidal>
```

```
          <bias>
            <float value="1" />
          </bias>
        </subtraction>
      </addition>
    </parameter>
    <parameter name="APE">
      <addition>
        <exponentiation>
          <sinusoidal>
            <float value="2" />
            <float value="4" />
            <float value="0" />
          </sinusoidal>
          <float value="2" />
        </exponentiation>
        <root>
          <addition>
            <exponentiation>
              <polynomialSampling xMin="1.0" xMax="10.0" step="1">
                <float value="1" />
                <float value="2" />
                <float value="1" />
                <float value="2" />
                <float value="1" />
                <float value="2" />
                <float value="1" />
                <float value="2" />
                <float value="1" />
                <float value="2" />
              </polynomialSampling>
              <float value="2" />
            </exponentiation>
            <exponentiation>
              <polynomialSampling xMin="1.0" xMax="10.0" step="1">
                <float value="1" />
                <float value="2" />
                <float value="1" />
                <float value="2" />
                <float value="1" />
                <float value="2" />
                <float value="1" />
                <float value="2" />
                <float value="1" />
                <float value="2" />
              </polynomialSampling>
              <float value="2" />
            </exponentiation>
          </addition>
          <float value="2" />
        </root>
      </addition>
  </parameter>
</errorsFile>
```

### 3.3.2. Process logic

This section shows the process logic of using the libraries in models source code.

Steps for using the Error Generation Libraries:

1. Include the OSFEG.h header file in your code

```
#include "OSFEG.h"
```

2. Create an instance of the ErrorSources class passing the name of the XML error definition file. The constructor throws an exception in case of error, so remind to catch it.

```
ErrorSources * reader = new ErrorSources(errorDefinitionFile);
```

3. Access the perturbation values by the complete name of the parameter and a double specifying the simulation step.

```
reader->getError(paramName, step);
```

4. Destroy the instance once not needed.

```
delete reader;
```

### 3.3.3. Examples of use

#### 3.3.3.1. C++ Programming Language

Here is an example of C++ code that uses the error generation libraries.

cppExample.cpp

```
#include "OSFEG.h"
#include <iostream>
#include <cstdlib>
#include <string>

using namespace std;

int main(int argc, char *argv[]) {
  cout << "Reading error sources definition file" << endl;
   try
   {
      string config(argv[1]);
      cout << "Reading error sources definition file " << config << endl;
      // Create an ErrorSources
      ErrorSources * reader = new ErrorSources(config);
      double step;
      string paramName = “Example Param Name”;
      reader->getError(paramName, step);
      delete reader;
   } catch (const runtime_error &e)
   {
      cerr << e.what() << endl;
   } catch (...)
   {
      cerr << argv[0] << " failed" << endl;
      exit(1);
   }
   exit(0);
}
```

# 4. OPENSF INTERFACES

This section gives a description of how to integrate external models into the openSF system.

Additionally, it is also described the mechanism to integrate third-party applications into the framework as well as user-configurable variables (Environment variables and help system interface).

## 4.1. Model Interfaces

As defined in section 1.4 a model is an openSF entity that represents a single executable file. This executable file represents the smallest component within a simulation chain, and it can implement a given scientific algorithm, instrument model, data processing or any desired part of the processing chain to be simulated within openSF.

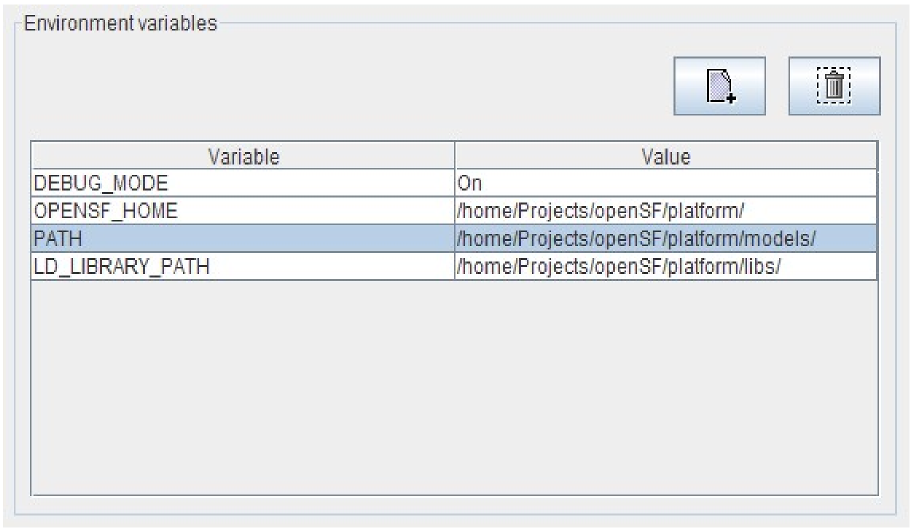
In order to integrate models into the framework, developers shall use an interface convention as described in ESA generic E2E simulator ICD [AD-E2E]. OpenSF is fully compliant with the interface definitions provided in this applicable document. In fact OSFI libraries can be seen as an implementation of the interfaces described.

Please refer to ESA generic E2E simulator ICD [AD-E2E] for a detailed description of the interfaces to apply when integrating models into openSF.

## 4.2. Advanced Interfaces

### 4.2.1. Environment variables

Users can define through the HMI a set of environment variables (associations between names and values) that will be set before the simulation execution. They will be active during the whole execution process launched by the HMI and not in other cases. In Figure 4-1 the interface for setting these environment variables is shown.


Environment variables

| Variable | Value |
|---|---|
| DEBUG_MODE | On |
| OPENSF_HOME | /home/Projects/openSF/platform/ |
| PATH | /home/Projects/openSF/platform/models/ |
| LD_LIBRARY_PATH | /home/Projects/openSF/platform/libs/ |

***Figure 4-1 Environment Variables Addition Interface***

One particular environment variable shall be set, OSFI_HOME. The location of the top-level directory on the local machine is stored by the HMI in this shell environment variable. This is the main mounting point for all the models, simulations and sessions.

A directory structure relative to OSFI_HOME is provided in section 4.2.3. Alternatively, users can also plug-in models that do not take advantage of this environment variable.

### 4.2.2. openSF Configuration Interfaces

This section describes the openSF interfaces for the configuration of user relevant variables in the openSF HMI. These configuration interfaces are not essential for the proper functioning of openSF system but can be useful for helping final system users.

#### 4.2.2.1. Help System

OpenSF help system is formed by a contextual tooltip system and a user accessible interface to the framework and system documentation. The access to system documentation mentioned before is based in the use of the Java Swing Jtree technology for the user interface and the integration of third party tools for documents visualisation.

OpenSF brings the user a mechanism to add new documents to this help system and the possibility of changing the way it is shown in the HMI through and XML file. This file describes the system documentation tree and can be edited by the user with its preferred text editor.

XML help file is located in the installation path of openSF with the name *helpSystem.xml.* The structure of the file is as follows:

- Root tag: *<Documentation>*
- Hierarchy tag: *<tagName>*, this "tagName" will be displayed as a folder in the user interface. The number of hierarchy tags is not limited.
- Document element: *<document name="name" type="type" value="file_name"/>*. This element is displayed as a leaf of the parent hierarchy tag.

| **Attribute Name** | **Value** |
|---|---|
| name | Desired displayed name for the document. Example: OpenSF Interface Control Document |
| type | Format of the file. Currently supported "pdf" and "html". |
| value | Complete name of the document file. |

An example of the help system file is shown below.

***helpSystem.xml***

```
<?xml version="1.0" encoding="UTF-8"?>
<Documentation>
  <Project_Name>
    <document name="Project ADD" type="pdf" value="Project -ADD.pdf"/>
  </Project_Name >
  <openSF>
    <platform>
      <document name="openSF ICD" type="pdf" value="openSF-ICD.pdf"/>
      <document name="openSF SUM" type="pdf" value="openSF-SUM.pdf"/>
    </platform>

    <OSFI>
      <document name="OSFI Developers Manual" type="pdf" value="openSF-DMS-
OSFI-DM-011.pdf"/>
    </OSFI>
  </openSF>
</Documentation>
```

The place where documentation files shall be stored is the *"help"* folder, located in the openSF root folder (see section 4.2.3).

Figure 4-2 shows the user interface correspondent to the help definition file listed above. The system tools used to open documentation files shall be specified through the openSF configuration system. This mechanism is described in [AD-SUM].

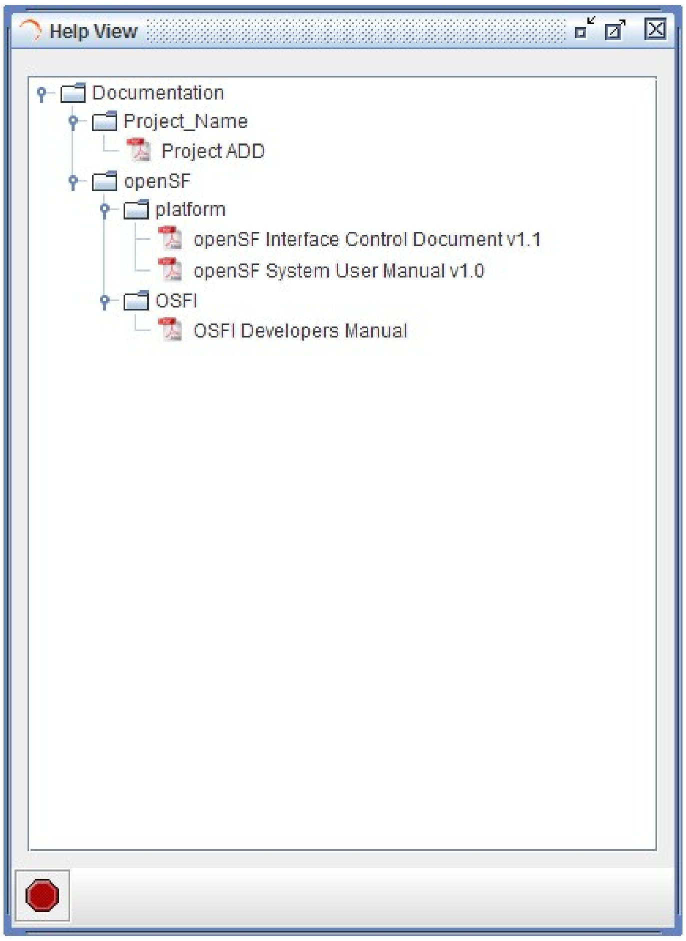


*Figure 4-2 Help View User-Interface*

#### 4.2.2.2. System Configuration

Properties file is the most common mechanism used by Java programs to configure the interface-related variables decoupling the source code from the application variables. This mechanism allows the easy maintenance, adaptation and localization of a Java application.

The Java properties file consists in a list of keys with a value associated. These keys are read by openSF application during the system execution to

This file shall be managed by openSF development team but advanced and experienced users can take advantages editing it in an offline way with a simple text editor.

```
openSF.acrobatBin=C\:/Program Files/Adobe/Acrobat 7.0/Acrobat/Acrobat
addToolToSession.SmallIcon=/mmi/presentation/images/Add24.gif
```

### 4.2.3. Folder structure

The table indicates the folder structure of the openSF system and the preferred placeholder for third-party components like models, tools, products or auxiliary files.

In case a third-party adopts a different folder structure OS's have the option to create links/shortcuts to the actual folder location. This folder structure is only a suggestion for a correct deployment of a whole simulation environment. Users are able to choose other structure in order to reach their simulation goals.

*Table 4-1: openSF folder structure and contents*

| Folder name (indented) | Contents |
|---|---|
| ❑ . | openSF home root<br>❑ "install". Scripts for setup in Linux.<br>❑ "openSF". Starting-up script for Linux.<br>❑ "openSF.properties". The configuration file.<br>❑ "openSFdb.sql". The original MySQL database script.<br>❑ "lgpl-3.0". Licensing scheme file |
| ❑ auxiliary | Auxiliary files for the framework |
| ❑ bin | Binary files of the framework. |
| ❑ help | Framework and system documentation |
| ❑ models | Placeholder for openSF models |
|   ❑ auxiliary | Auxiliary files for the models |
|   ❑ common | Common sources, modules and utilities used by the models |
|   ❑ <model_name> | Model root. Every model has one directory structured as this one. Model folders might include additional folders. |
|     ❑ aux | Auxiliary files |
|     ❑ bin | Binary files |
|     ❑ conf | Configuration files |
|     ❑ src | Sources |
| ❑ products | Some product file samples |
| ❑ sessions | OpenSF sessions root folder |
|   ❑ <session_name> | Session folder. Every session, once executed, has one directory structured as this one.<br>This folder will have the session script and, if generated, the session report. |
| ❑ src | Framework sources |
| ❑ tmp | Framework temporary directory |
| ❑ tools | Placeholder for openSF tools |

## 4.2.4. XML file format for defining openSF elements externally

OpenSF elements can be configured externally using XML import files (as defined in [XML]) formatted with the following elements (refer to Figure 4-3):

- Element “<descriptors>” and “<ios>”: encloses the model descriptors and their file counterparts;
- Element “<tools>”: specifies an association of file extensions to external executable tools;
- Element “<stages>”: describes the available model stages;
- Element “<models>”: defines the application models;
- Element “<simulations>”: defines the set of simulations available;
- Element “<modelofsimulations>”: defines the set of model association to simulations;
- Element “<sessions>”: specifies a set of (not yet executed) sessions;
- Element “<simulationofsessions>”: specifies a set of simulations association to sessions;
- Element “<paramofsessions>”: specifies a set of parameters association to sessions;
- Element “<ioofsessions>”: specifies a set of descriptors association with sessions;
- Element “<toolofsessions>”: specifies a set of tool association with sessions;
- Element “<timeofsessions>”: specifies a set of execution times association with sessions;
- Element “<modelperturbation>”: specifies the definition of model perturbations;

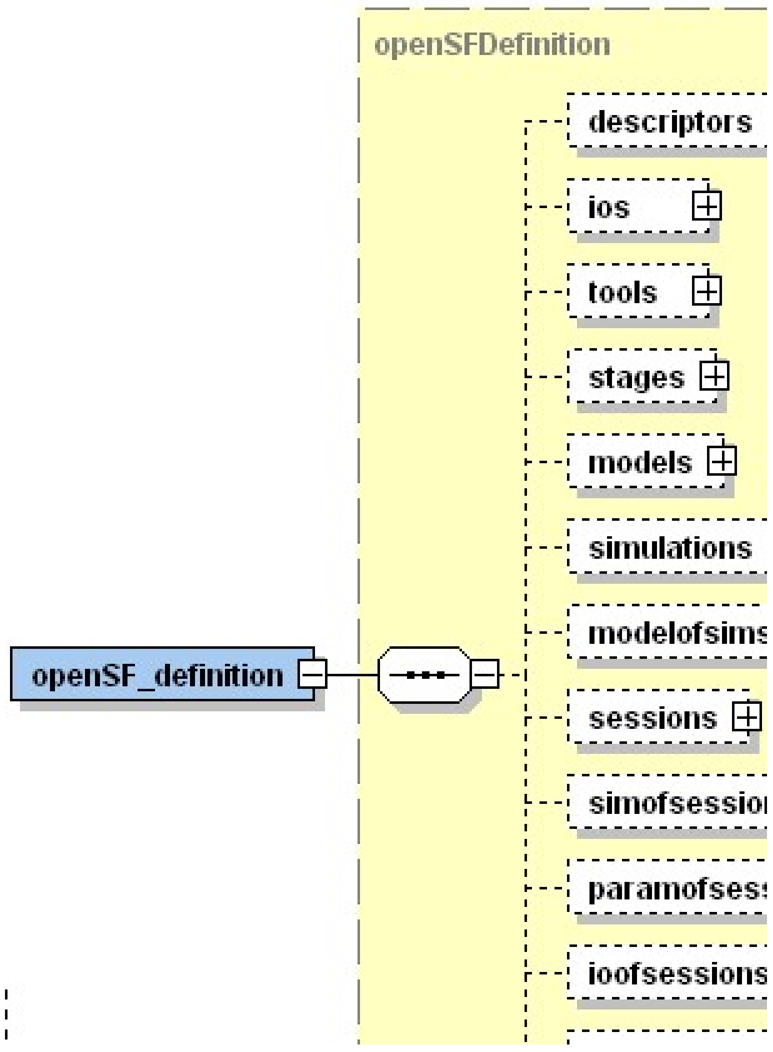


*Figure 4-3 Top level XSD schema for defining openSF element thru XML*

The XSD schema defines the types related with the several data entries, namely the data types and the size of the fields; most importantly the schema also enforces the constraints to apply to the several entries, namely regarding uniqueness of key values. It should be noted that the XML adopted version is 1.1 in order to use facets as xs:assert and xs:alternatives in defining the data constraints. Furthermore the data format was specified so that the format used when importing data to openSF is compatible with the format to use when exporting data from openSF. The following figures give more detail regarding the definition of the several elements that can be configures externally. Additionally the XSD source is available in “ANNEX A: XSD Schema for openSF elements” along with an example of XML import file.

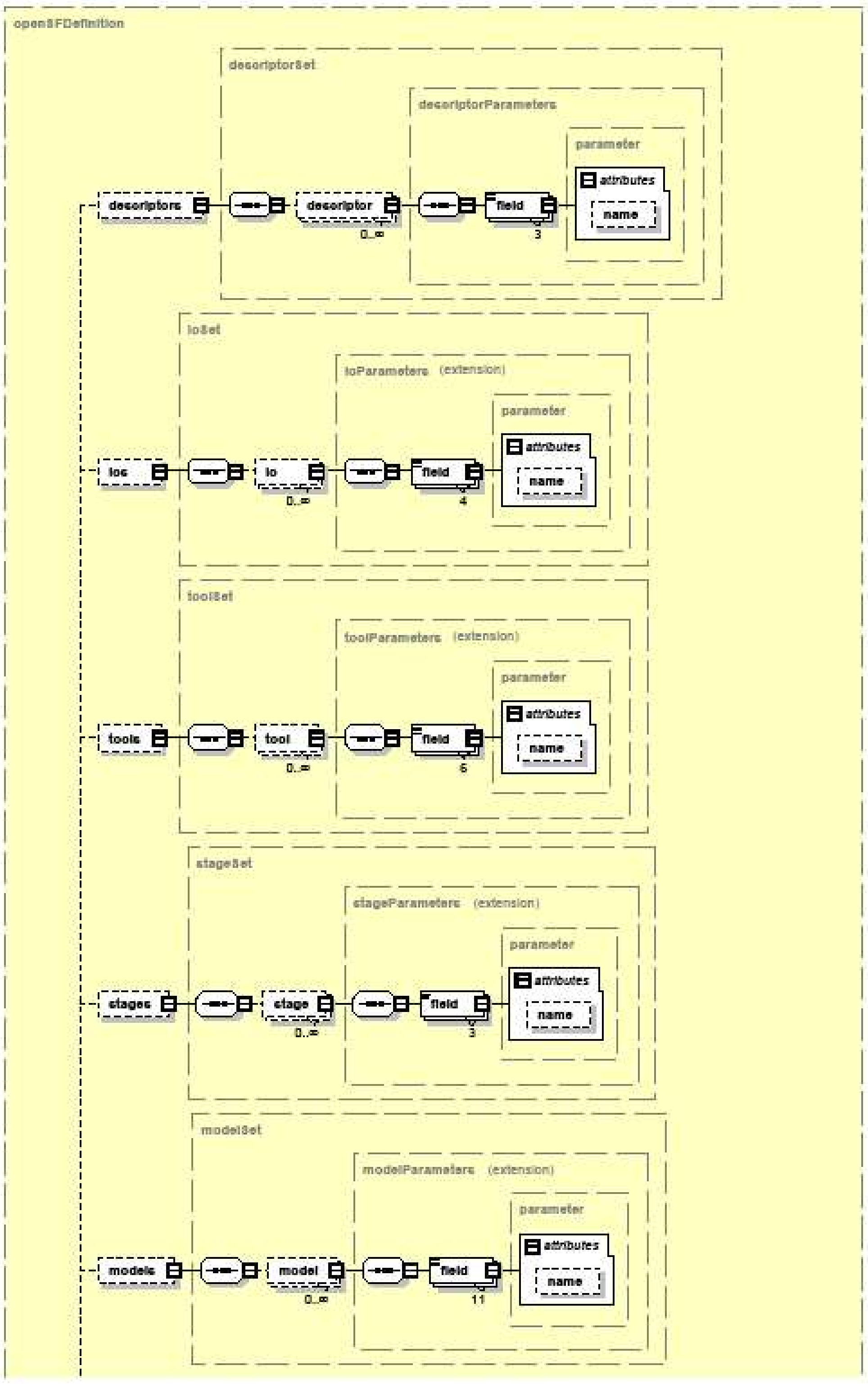


*Figure 4-4 Detailed XSD schema for defining openSF element thru XML (part 1)*

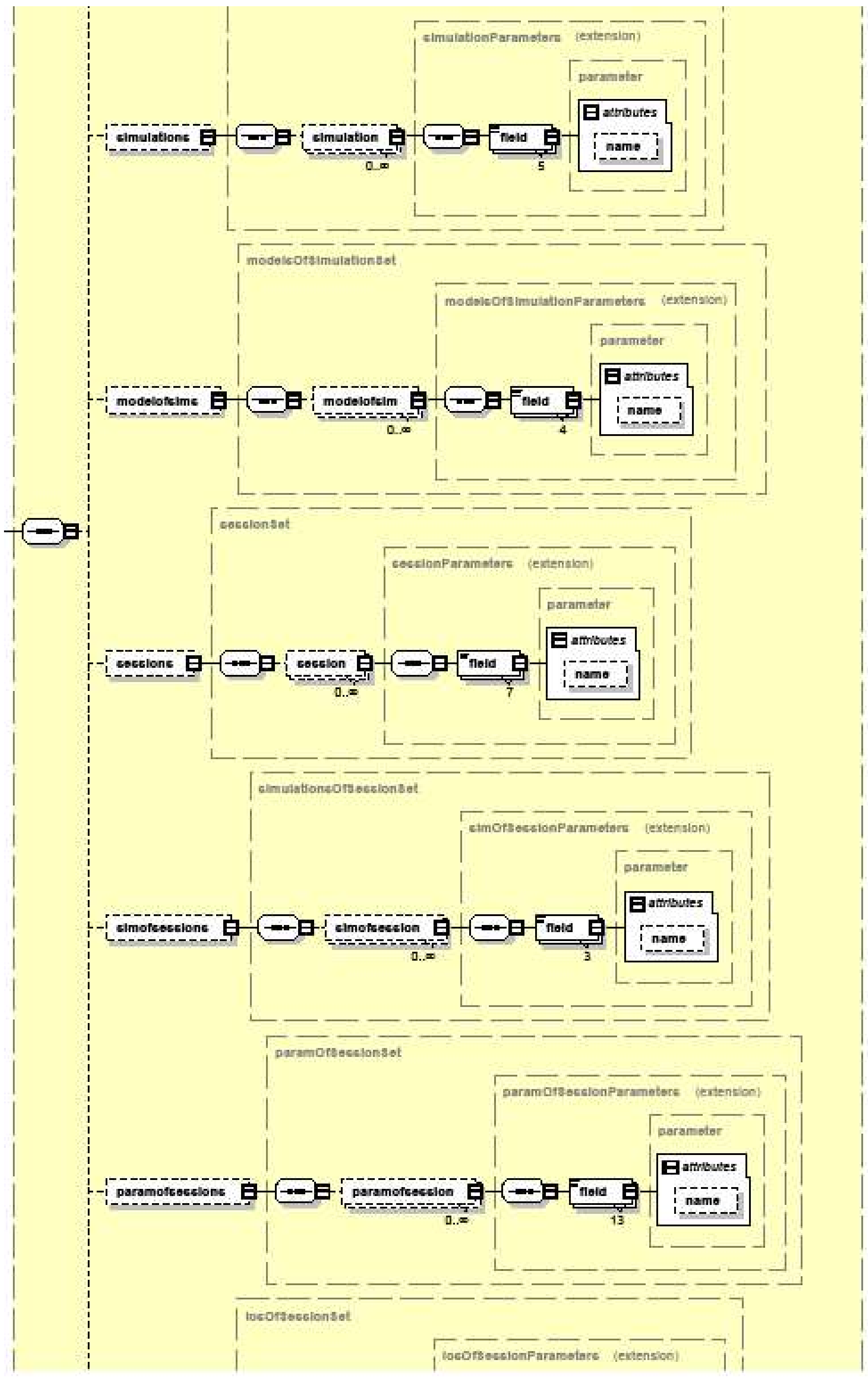


*Figure 4-5 Detailed XSD schema for defining openSF element thru XML (part 2)*

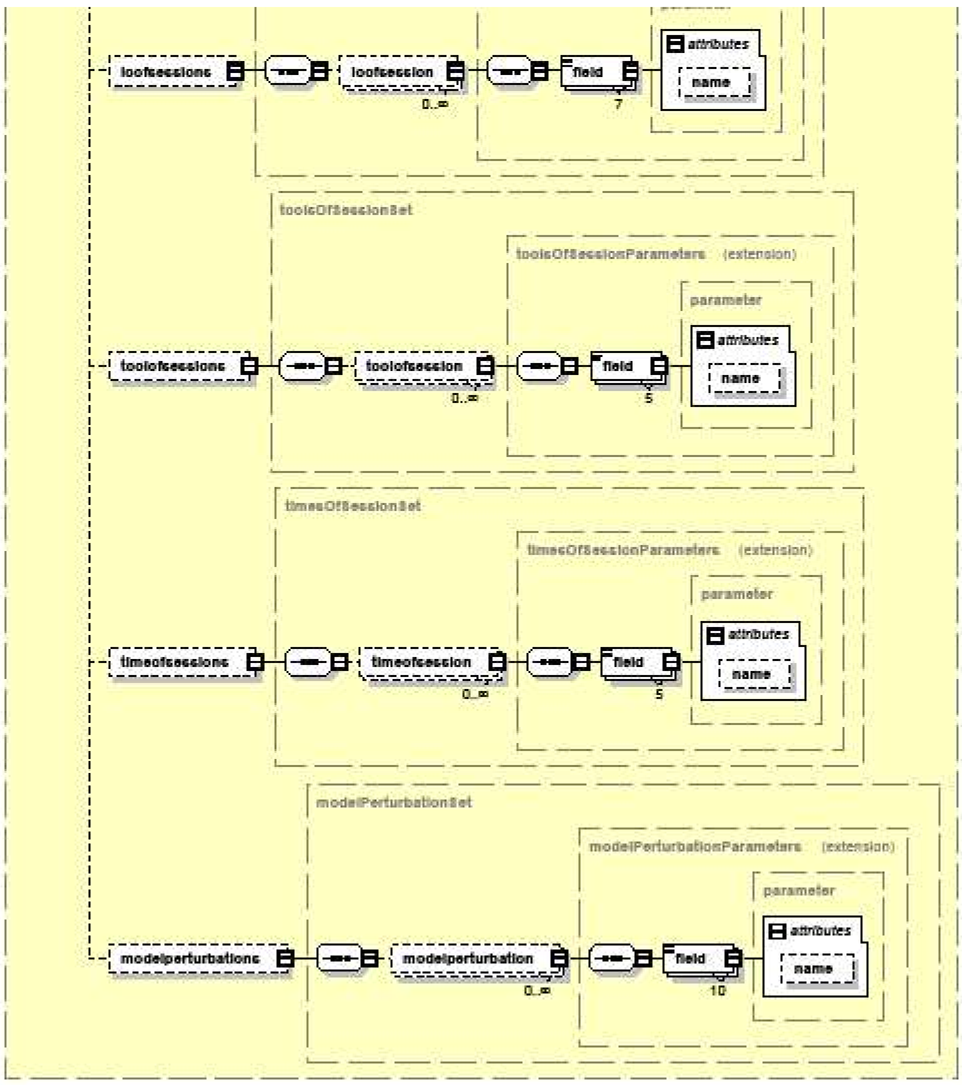


*Figure 4-6 Detailed XSD schema for defining openSF element thru XML (part 3)*

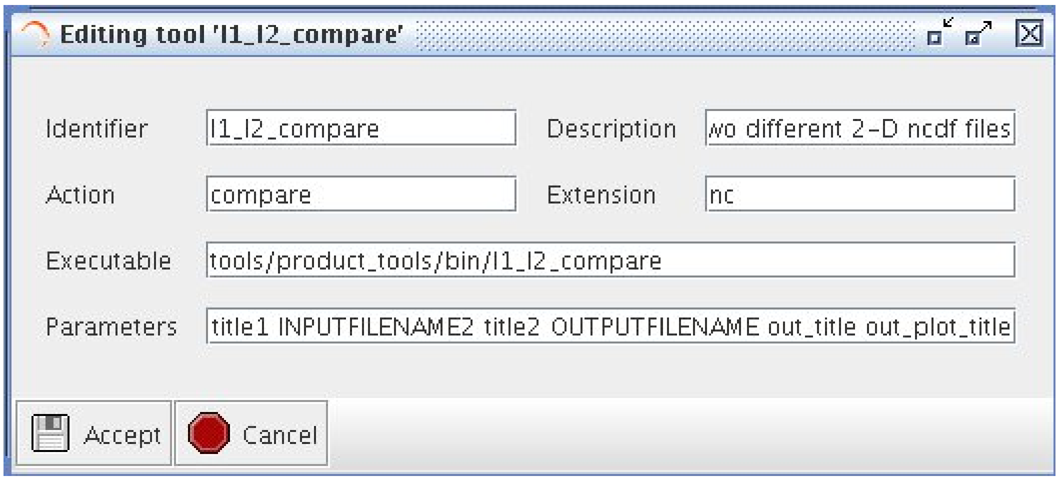
## 4.3. Third-party tool Interface

A third party tool as defined in section 1.4 is any software application that can be executed within the framework.

These tools can be simulation-dedicated applications (exclusively made for the desired project) or user preferred third-party applications such as viewers, editors, browsers etc… (Matlab, Firefox, Acrobat, GIS, Notepad). When the application is linked to a simulation output or auxiliary file it can also be called product tool, as it is associated to a file, or set of files, generated by the simulation chain.

The only constraint for these tools is that they must allow the command line invocation, as it is the mechanism used by the framework to launch this kind of applications[2].

OpenSF allows the association of tools to a file extension, so users are able to launch these tools by the use of a contextual menu in the openSF file-system view. The framework also allows to automatically executing them any time a simulation is executed (session). In Figure 4-7 the user interface for tool addition is shown.


Editing tool 'l1_l2_compare'

Identifier: l1_l2_compare
Description: wo different 2-D ncdf files
Action: compare
Extension: nc
Executable: tools/product_tools/bin/l1_l2_compare
Parameters: title1 INPUTFILENAME2 title2 OUTPUTFILENAME out_title out_plot_title

*Figure 4-7 User interface for tool addition*

The mechanism provided for adding a tool to the framework will be described in [AD-SUM]. In this document it can also be found details on how to link these tools to the simulation-related files.

[2] Almost all software applications allow command line invocation.

# 5. ANNEX A: XSD SCHEMA FOR OPENSF ELEMENTS

This annex contains the XSD schema that defines the XML syntax for specifying openSF elements externally to the application. The source files for these schemas can be found in appendix to this document along with an example of XML describing the openSF application specifications used for Validation purposes.

openSFImport.xsd:

```
<?xml version="1.1" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.1">
   <xs:annotation>
      <xs:documentation>$LastChangedDate: $ $LastChangedRevision:
$</xs:documentation>
   </xs:annotation>
   <xs:include schemaLocation="openSFImportTypes.xsd"/>
   <xs:element name="openSF_definition" type="openSFDefinition"/>
</xs:schema>
```

openSFImportTypes.xsd:

```
<?xml version="1.1" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.1">
   <xs:annotation>
      <xs:documentation>$LastChangedDate: $ $LastChangedRevision:
$</xs:documentation>
   </xs:annotation>
   <!-- Concepts parameters -->
   <xs:complexType name="descriptorParameters">
      <xs:sequence>
         <xs:element name="field" type="parameter" minOccurs="3" maxOccurs="3">
            <xs:alternative test="@name='id'" type="shortKeyParameter"/>
            <xs:alternative test="@name='description'"
type="unrestrictedStringParameter"/>
            <xs:alternative test="@name='noFiles'" type="shortNumberParameter"/>
            <xs:alternative type="xs:error"/>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="ioParameters">
      <xs:sequence>
         <xs:element name="field" type="parameter" minOccurs="4" maxOccurs="4">
            <xs:alternative test="@name='descriptorId'" type="longKeyParameter"/>
            <xs:alternative test="@name='place'" type="shortNumberParameter"/>
            <xs:alternative test="@name='id'" type="unrestrictedStringParameter"/>
            <xs:alternative test="@name='description'"
type="unrestrictedStringParameter"/>
            <xs:alternative type="xs:error"/>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="stageParameters">
      <xs:sequence>
      <xs:element name="field" type="parameter" minOccurs="3" maxOccurs="3">
        <xs:alternative test="@name='id'" type="shortKeyParameter"/>
        <xs:alternative test="@name='description'"
type="unrestrictedStringParameter"/>
        <xs:alternative test="@name='position'" type="shortNumberParameter"/>
        <xs:alternative type="xs:error"/>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="modelParameters">
    <xs:sequence>
      <xs:element name="field" type="parameter" minOccurs="11" maxOccurs="11">
        <xs:alternative test="@name='id'" type="shortKeyParameter"/>
        <xs:alternative test="@name='version'" type="decimalParameter"/>
        <xs:alternative test="@name='description'"
type="unrestrictedStringParameter"/>
        <xs:alternative test="@name='author'" type="stringParameter"/>
        <xs:alternative test="@name='type'" type="shortStringParameter"/>
        <xs:alternative test="@name='sourceFile'"
type="unrestrictedStringParameter"/>
        <xs:alternative test="@name='binaryFile'" type="longStringParameter"/>
        <xs:alternative test="@name='confSchema'"
type="unrestrictedStringParameter"/>
        <xs:alternative test="@name='defaultConfFile'"
type="unrestrictedStringParameter"/>
        <xs:alternative test="@name='inputId'" type="shortStringParameter"/>
        <xs:alternative test="@name='outputId'" type="shortStringParameter"/>
        <xs:alternative type="xs:error"/>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="simulationParameters">
    <xs:sequence>
      <xs:element name="field" type="parameter" minOccurs="5" maxOccurs="5">
        <xs:alternative test="@name='id'" type="shortKeyParameter"/>
        <xs:alternative test="@name='description'"
type="unrestrictedStringParameter"/>
        <xs:alternative test="@name='author'" type="stringParameter"/>
        <xs:alternative test="@name='start'" type="shortStringParameter"/>
        <xs:alternative test="@name='end'" type="shortStringParameter"/>
        <xs:alternative type="xs:error"/>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="modelsOfSimulationParameters">
    <xs:sequence>
      <xs:element name="field" type="parameter" minOccurs="4" maxOccurs="4">
        <xs:alternative test="@name='simId'" type="longKeyParameter"/>
        <xs:alternative test="@name='place'" type="shortNumberParameter"/>
        <xs:alternative test="@name='modelId'" type="longKeyParameter"/>
        <xs:alternative test="@name='modelVersion'" type="decimalParameter"/>
        <xs:alternative type="xs:error"/>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="sessionParameters">
    <xs:sequence>
```

```
      <xs:element name="field" type="parameter" minOccurs="7" maxOccurs="7">
        <xs:alternative test="@name='id'" type="longKeyParameter">
          <!-- sessionName (YYYY-MM-DD hh:mm:ss.SSS) -->
        </xs:alternative>
        <xs:alternative test="@name='description'"
type="unrestrictedStringParameter"/>
        <xs:alternative test="@name='author'" type="stringParameter"/>
        <xs:alternative test="@name='status'" type="stringParameter"/>
        <xs:alternative test="@name='lastModel'"
type="extendedShortNumberParameter"/>
        <xs:alternative test="@name='breakpoints'" type="numberListParameter">
          <!-- List of breakpoints as model indexes -->
        </xs:alternative>
        <xs:alternative test="@name='duration'" type="longNumberParameter">
          <!-- Duration since dateTime to the end of the session (in
milliseconds) -->
        </xs:alternative>
        <xs:alternative type="xs:error"/>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="simOfSessionParameters">
    <xs:sequence>
      <xs:element name="field" type="parameter" minOccurs="3" maxOccurs="3">
        <xs:alternative test="@name='sessionId'" type="longKeyParameter">
          <!-- sessionName (YYYY-MM-DD hh:mm:ss.SSS) -->
        </xs:alternative>
        <xs:alternative test="@name='place'" type="shortNumberParameter"/>
        <xs:alternative test="@name='simId'" type="shortKeyParameter"/>
        <xs:alternative type="xs:error"/>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="paramOfSessionParameters">
    <xs:sequence>
      <xs:element name="field" type="parameter" minOccurs="13" maxOccurs="13">
        <xs:alternative test="@name='id'" type="longStringParameter"/>
        <xs:alternative test="@name='sessionId'" type="longKeyParameter"/>
        <xs:alternative test="@name='simPlace'"
type="extendedShortNumberParameter"/>
        <xs:alternative test="@name='modelPlace'"
type="extendedShortNumberParameter"/>
        <xs:alternative test="@name='type'" type="shortKeyParameter"/>
        <xs:alternative test="@name='description'"
type="unrestrictedStringParameter"/>
        <xs:alternative test="@name='units'"
type="unrestrictedShortStringParameter"/>
        <xs:alternative test="@name='xmin'"
type="unrestrictedStringParameter"/>
        <xs:alternative test="@name='xmax'"
type="unrestrictedStringParameter"/>
        <xs:alternative test="@name='dims'"
type="unrestrictedShortStringParameter"/>
        <xs:alternative test="@name='valuesList'" type="textParameter"/>
        <xs:alternative test="@name='paramPlace'"
type="shortNumberParameter"/>
        <xs:alternative type="xs:error"/>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="toolParameters">
    <xs:sequence>
      <xs:element name="field" type="parameter" minOccurs="6" maxOccurs="6">
        <xs:alternative test="@name='id'" type="longKeyParameter"/>
        <xs:alternative test="@name='action'" type="stringParameter"/>
        <xs:alternative test="@name='description'"
type="unrestrictedStringParameter"/>
        <xs:alternative test="@name='extension'" type="longStringParameter"/>
        <xs:alternative test="@name='executable'"
type="unrestrictedStringParameter"/>
        <xs:alternative test="@name='param'"
type="unrestrictedStringParameter"/>
        <xs:alternative type="xs:error"/>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="toolsOfSessionParameters">
    <xs:sequence>
      <xs:element name="field" type="parameter" minOccurs="5" maxOccurs="5">
        <xs:alternative test="@name='id'" type="longKeyParameter"/>
        <xs:alternative test="@name='action'" type="stringParameter"/>
        <xs:alternative test="@name='sessionId'" type="longKeyParameter"/>
        <xs:alternative test="@name='place'" type="shortNumberParameter"/>
        <xs:alternative test="@name='parameter'"
type="unrestrictedStringParameter"/>
        <xs:alternative type="xs:error"/>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="iosOfSessionParameters">
    <xs:sequence>
      <xs:element name="field" type="parameter" minOccurs="7" maxOccurs="7">
        <xs:alternative test="@name='sessionId'" type="longKeyParameter"/>
        <xs:alternative test="@name='simPlace'"
type="extendedShortNumberParameter"/>
        <xs:alternative test="@name='modelPlace'"
type="extendedShortNumberParameter"/>
        <xs:alternative test="@name='ioPlace'"
type="extendedShortNumberParameter"/>
        <xs:alternative test="@name='type'"
type="unrestrictedShortStringParameter"/>
        <xs:alternative test="@name='location'"
type="unrestrictedStringParameter"/>
        <xs:alternative test="@name='source'"
type="unrestrictedStringParameter"/>
        <xs:alternative type="xs:error"/>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="timesOfSessionParameters">
    <xs:sequence>
      <xs:element name="field" type="parameter" minOccurs="5" maxOccurs="5">
        <xs:alternative test="@name='sessionid'" type="longKeyParameter"/>
        <xs:alternative test="@name='modelid'" type="longKeyParameter"/>
        <xs:alternative test="@name='modelVersion'" type="decimalParameter"/>
        <xs:alternative test="@name='modelplace'"
type="shortNumberParameter"/>
        <xs:alternative test="@name='time'" type="longNumberParameter"/>
        <xs:alternative type="xs:error"/>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="modelPerturbationParameters">
    <xs:sequence>
      <xs:element name="field" type="parameter" minOccurs="10" maxOccurs="10">
        <xs:alternative test="@name='sessionId'" type="longKeyParameter"/>
        <xs:alternative test="@name='simId'" type="longKeyParameter"/>
        <xs:alternative test="@name='simPlace'"
type="extendedShortNumberParameter"/>
        <xs:alternative test="@name='modelId'" type="longKeyParameter"/>
        <xs:alternative test="@name='modelVersion'" type="decimalParameter"/>
        <xs:alternative test="@name='description'"
type="unrestrictedStringParameter"/>
        <xs:alternative test="@name='nshots'" type="shortNumberParameter"/>
        <xs:alternative test="@name='tMin'" type="doubleParameter"/>
        <xs:alternative test="@name='tMax'" type="doubleParameter"/>
        <xs:alternative test="@name='file'"
type="unrestrictedStringParameter"/>
        <xs:alternative type="xs:error"/>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <!-- Sets of parameters -->
  <xs:complexType name="descriptorSet">
    <xs:sequence>
      <xs:element name="descriptor" type="descriptorParameters" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <!-- Primary key -->
    <xs:assert test="every $descriptor in descriptor[position() lt last()]
satisfies (every $other in $descriptor/following-sibling::descriptor satisfies
(not(data($other/field[@name = 'id']) eq data($descriptor/field[@name =
'id'])))))" />
    <!-- xs:key name="descriptorKey">
         <xs:selector xpath="descriptor"/>
         <xs:field xpath="id"/>
    </xs:key -->
  </xs:complexType>
  <xs:complexType name="ioSet">
    <xs:sequence>
      <xs:element name="io" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:complexContent>
            <xs:extension base="ioParameters"/>
          </xs:complexContent>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <!-- Primary key -->
    <xs:assert test="every $io in io[position() lt last()] satisfies (every
$other in $io/following-sibling::io satisfies (not(data($other/field[@name =
'descriptorId']) eq data($io/field[@name = 'descriptorId'])) or
not(data($other/field[@name = 'place']) eq data($io/field[@name = 'place']))))"
/>
    <!-- xs:key name="ioKey">
         <xs:selector xpath="io"/>
         <xs:field xpath="descriptorId"/>
         <xs:field xpath="place"/>
    </xs:key -->
  </xs:complexType>
  <xs:complexType name="stageSet">
    <xs:sequence>
      <xs:element name="stage" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:complexContent>
            <xs:extension base="stageParameters"/>
          </xs:complexContent>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <!-- Primary key -->
    <xs:assert test="every $stage in stage[position() lt last()] satisfies
(every $other in $stage/following-sibling::stage satisfies
(not(data($other/field[@name = 'position']) eq data($stage/field[@name =
'position']))))" />
    <!-- xs:key name="stageKey">
         <xs:selector xpath="stage"/>
         <xs:field xpath="position"/>
    </xs:key -->
  </xs:complexType>
  <xs:complexType name="toolSet">
    <xs:sequence>
      <xs:element name="tool" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:complexContent>
            <xs:extension base="toolParameters"/>
          </xs:complexContent>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <!-- Primary Key -->
    <xs:assert test="every $tool in tool[position() lt last()] satisfies (every
$other in $tool/following-sibling::tool satisfies (not(data($other/field[@name =
'id']) eq data($tool/field[@name = 'id'])) or not(data($other/field[@name =
'action']) eq data($tool/field[@name = 'action'])) or
not(data($other/field[@name = 'extension']) eq data($tool/field[@name =
'extension']))))" />
    <!-- xs:key name="toolKey">
       <xs:selector xpath="tool"/>
       <xs:field xpath="id"/>
       <xs:field xpath="action"/>
       <xs:field xpath="extension"/>
    </xs:key -->
  </xs:complexType>
  <xs:complexType name="modelSet">
    <xs:sequence>
      <xs:element name="model" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
```

```
              <xs:complexContent>
                <xs:extension base="modelParameters"/>
              </xs:complexContent>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
        <!-- Primary Key -->
        <xs:assert test="every $model in model[position() lt last()] satisfies
(every $other in $model/following-sibling::model satisfies
(not(data($other/field[@name = 'id']) eq data($model/field[@name = 'id'])) or
not(data($other/field[@name = 'version']) eq data($model/field[@name =
'version']))))" />
        <!-- xs:key name="modelKey">
             <xs:selector xpath="model"/>
             <xs:field xpath="id"/>
             <xs:field xpath="version"/>
        </xs:key -->
      </xs:complexType>
      <xs:complexType name="simulationSet">
        <xs:sequence>
          <xs:element name="simulation" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:complexContent>
                <xs:extension base="simulationParameters"/>
              </xs:complexContent>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
        <!-- Primary key -->
        <xs:assert test="every $sim in simulation[position() lt last()] satisfies
(every $other in $sim/following-sibling::simulation satisfies
(not(data($other/field[@name = 'id']) eq data($sim/field[@name = 'id']))))" />
        <!-- xs:key name="simulationKey">
             <xs:selector xpath="simulation"/>
             <xs:field xpath="id"/>
        </xs:key -->
      </xs:complexType>
      <xs:complexType name="modelsOfSimulationSet">
        <xs:sequence>
          <xs:element name="modelofsim" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:complexContent>
                <xs:extension base="modelsOfSimulationParameters"/>
              </xs:complexContent>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
        <!-- Primary Key -->
        <xs:assert test="every $model in modelofsim[position() lt last()] satisfies
(every $other in $model/following-sibling::modelofsim satisfies
(not(data($other/field[@name = 'simId']) eq data($model/field[@name = 'simId']))
or not(data($other/field[@name = 'place']) eq data($model/field[@name =
'place']))))" />
        <!-- xs:key name="modelsOfSimulationKey">
             <xs:selector xpath="modelofsim"/>
             <xs:field xpath="simId"/>
             <xs:field xpath="place"/>
```

```
     </xs:key -->
   </xs:complexType>
   <xs:complexType name="sessionSet">
     <xs:sequence>
       <xs:element name="session" minOccurs="0" maxOccurs="unbounded">
         <xs:complexType>
           <xs:complexContent>
             <xs:extension base="sessionParameters"/>
           </xs:complexContent>
         </xs:complexType>
       </xs:element>
     </xs:sequence>
     <!-- Primary key -->
     <xs:assert test="every $session in session[position() lt last()] satisfies
(every $other in $session/following-sibling::session satisfies
(not(data($other/field[@name = 'id']) eq data($session/field[@name = 'id']))))"
/>
     <!-- xs:key name="sessionKey">
          <xs:selector xpath="session"/>
          <xs:field xpath="id"/>
           </xs:key -->
   </xs:complexType>
   <xs:complexType name="simulationsOfSessionSet">
     <xs:sequence>
       <xs:element name="simofsession" minOccurs="0" maxOccurs="unbounded">
         <xs:complexType>
           <xs:complexContent>
             <xs:extension base="simOfSessionParameters"/>
           </xs:complexContent>
         </xs:complexType>
       </xs:element>
     </xs:sequence>
     <!-- Primary Key -->
     <xs:assert test="every $sim in simofsession[position() lt last()] satisfies
(every $other in $sim/following-sibling::simofsession satisfies
(not(data($other/field[@name = 'sessionId']) eq data($sim/field[@name =
'sessionId'])) or not(data($other/field[@name = 'place']) eq
data($sim/field[@name = 'place']))))" />
     <!-- xs:key name="simulationsOfSessionKey">
          <xs:selector xpath="simofsession"/>
          <xs:field xpath="sessionId"/>
          <xs:field xpath="place"/>
     </xs:key -->
   </xs:complexType>
   <xs:complexType name="paramOfSessionSet">
     <xs:sequence>
       <xs:element name="paramofsession" minOccurs="0" maxOccurs="unbounded">
         <xs:complexType>
           <xs:complexContent>
             <xs:extension base="paramOfSessionParameters"/>
           </xs:complexContent>
         </xs:complexType>
       </xs:element>
     </xs:sequence>
     <!-- Primary Key -->
     <xs:assert test="every $param in paramofsession[position() lt last()]
satisfies (every $other in $param/following-sibling::paramofsession satisfies
(not(data($other/field[@name = 'sessionId']) eq data($param/field[@name =
'sessionId'])) or not(data($other/field[@name = 'simPlace']) eq
data($param/field[@name = 'simPlace'])) or not(data($other/field[@name =
'modelPlace']) eq data($param/field[@name = 'modelPlace'])) or
not(data($other/field[@name = 'paramPlace']) eq data($param/field[@name =
'paramPlace']))))" />
     <!-- xs:key name="paramOfSessionKey">
          <xs:selector xpath="paramofsession"/>
          <xs:field xpath="sessionId"/>
          <xs:field xpath="simPlace"/>
          <xs:field xpath="modelPlace"/>
          <xs:field xpath="paramPlace"/>
     </xs:key -->
   </xs:complexType>
   <xs:complexType name="toolsOfSessionSet">
     <xs:sequence>
       <xs:element name="toolofsession" minOccurs="0" maxOccurs="unbounded">
         <xs:complexType>
           <xs:complexContent>
             <xs:extension base="toolsOfSessionParameters"/>
           </xs:complexContent>
         </xs:complexType>
       </xs:element>
     </xs:sequence>
     <!-- Primary Key -->
     <xs:assert test="every $tool in toolofsession[position() lt last()]
satisfies (every $other in $tool/following-sibling::toolofsession satisfies
(not(data($other/field[@name = 'id']) eq data($tool/field[@name = 'id'])) or
not(data($other/field[@name = 'action']) eq data($tool/field[@name = 'action']))
or not(data($other/field[@name = 'sessionId']) eq data($tool/field[@name =
'sessionId'])) or not(data($other/field[@name = 'place']) eq
data($tool/field[@name = 'place']))))" />
     <!-- xs:key name="toolsOfSessionKey">
          <xs:selector xpath="toolofsession"/>
          <xs:field xpath="id"/>
          <xs:field xpath="action"/>
          <xs:field xpath="sessionId"/>
          <xs:field xpath="place"/>
     </xs:key -->
   </xs:complexType>
   <xs:complexType name="iosOfSessionSet">
     <xs:sequence>
       <xs:element name="ioofsession" minOccurs="0" maxOccurs="unbounded">
         <xs:complexType>
           <xs:complexContent>
             <xs:extension base="iosOfSessionParameters"/>
           </xs:complexContent>
         </xs:complexType>
       </xs:element>
     </xs:sequence>
     <!-- Primary Key -->
     <xs:assert test="every $io in ioofsession[position() lt last()] satisfies
(every $other in $io/following-sibling::ioofsession satisfies
(not(data($other/field[@name = 'sessionId']) eq data($io/field[@name =
'sessionId'])) or not(data($other/field[@name = 'simPlace']) eq
data($io/field[@name = 'simPlace'])) or not(data($other/field[@name =
'modelPlace']) eq data($io/field[@name = 'modelPlace'])) or
not(data($other/field[@name = 'ioPlace']) eq data($io/field[@name = 'ioPlace']))
or not(data($other/field[@name = 'type']) eq data($io/field[@name = 'type']))))"
/>
     <!-- xs:key name="iosOfSessionKey">
          <xs:selector xpath="ioofsession"/>
          <xs:field xpath="sessionId"/>
          <xs:field xpath="simPlace"/>
          <xs:field xpath="modelPlace"/>
          <xs:field xpath="ioPlace"/>
          <xs:field xpath="type"/>
     </xs:key -->
   </xs:complexType>
   <xs:complexType name="timesOfSessionSet">
     <xs:sequence>
       <xs:element name="timeofsession" minOccurs="0" maxOccurs="unbounded">
         <xs:complexType>
           <xs:complexContent>
             <xs:extension base="timesOfSessionParameters"/>
           </xs:complexContent>
         </xs:complexType>
       </xs:element>
     </xs:sequence>
     <!-- Primary Key -->
     <xs:assert test="every $time in timeofsession[position() lt last()]
satisfies (every $other in $time/following-sibling::timeofsession satisfies
(not(data($other/field[@name = 'sessionid']) eq data($time/field[@name =
'sessionid'])) or not(data($other/field[@name = 'modelplace']) eq
data($time/field[@name = 'modelplace']))))" />
     <!-- xs:key name="toolsOfSessionKey">
          <xs:selector xpath="timeofsession"/>
          <xs:field xpath="sessionid"/>
          <xs:field xpath="modelplace"/>
     </xs:key -->
   </xs:complexType>
   <xs:complexType name="modelPerturbationSet">
     <xs:sequence>
       <xs:element name="modelperturbation" minOccurs="0"
maxOccurs="unbounded">
         <xs:complexType>
           <xs:complexContent>
             <xs:extension base="modelPerturbationParameters"/>
           </xs:complexContent>
         </xs:complexType>
       </xs:element>
     </xs:sequence>
     <!-- Primary Key -->
     <xs:assert test="every $perturbation in modelperturbation[position() lt
last()] satisfies (every $other in $perturbation/following-
sibling::modelperturbation satisfies (not(data($other/field[@name =
'sessionId']) eq data($perturbation/field[@name = 'sessionId'])) or
not(data($other/field[@name = 'simPlace']) eq data($perturbation/field[@name =
'simPlace'])) or not(data($other/field[@name = 'modelId']) eq
data($perturbation/field[@name = 'modelId']))))" />
     <!-- xs:key name="modelPerturbationKey">
       <xs:selector xpath="modelPerturbation"/>
       <xs:field xpath="sessionId"/>
       <xs:field xpath="simPlace"/>
```

```
      <xs:field xpath="modelId"/>
    </xs:key -->
  </xs:complexType>
  <!-- Basic types -->
  <xs:complexType name="parameter">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="name" type="xs:string"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <!-- id Parameter -->
  <xs:complexType name="shortKeyParameter">
    <xs:simpleContent>
      <xs:restriction base="parameter">
        <xs:pattern value="[^ ]{1,25}"/>
      </xs:restriction>
    </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="longKeyParameter">
    <xs:simpleContent>
      <xs:restriction base="parameter">
        <xs:pattern value="[^ ]{1,75}"/>
      </xs:restriction>
    </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="shortStringParameter">
    <xs:simpleContent>
      <xs:restriction base="parameter">
        <xs:minLength value="1"/>
        <xs:maxLength value="25"/>
        <xs:pattern value="[A-Za-z0-9].{0,24}"/>
      </xs:restriction>
    </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="stringParameter">
    <xs:simpleContent>
      <xs:restriction base="parameter">
        <xs:minLength value="1"/>
        <xs:maxLength value="75"/>
        <xs:pattern value="[A-Za-z0-9].{0,74}"/>
      </xs:restriction>
    </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="longStringParameter">
    <xs:simpleContent>
      <xs:restriction base="parameter">
        <xs:minLength value="1"/>
        <xs:maxLength value="255"/>
        <xs:pattern value="[A-Za-z0-9].{0,254}"/>
      </xs:restriction>
    </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="unrestrictedStringParameter">
    <xs:simpleContent>
      <xs:restriction base="parameter">
        <xs:minLength value="0"/>
```

```
            <xs:maxLength value="255"/>
         </xs:restriction>
      </xs:simpleContent>
   </xs:complexType>
   <xs:complexType name="unrestrictedShortStringParameter">
      <xs:simpleContent>
         <xs:restriction base="parameter">
            <xs:minLength value="0"/>
            <xs:maxLength value="25"/>
         </xs:restriction>
      </xs:simpleContent>
   </xs:complexType>
   <xs:complexType name="textParameter">
      <xs:simpleContent>
         <xs:restriction base="parameter">
            <xs:minLength value="0"/>
            <xs:maxLength value="1000"/>
         </xs:restriction>
      </xs:simpleContent>
   </xs:complexType>
   <xs:complexType name="numberListParameter">
      <xs:simpleContent>
         <xs:restriction base="parameter">
            <xs:minLength value="0"/>
            <xs:maxLength value="25"/>
            <xs:pattern value="[(\d+)(,\s*\d+)*]{0,1}"/>
         </xs:restriction>
      </xs:simpleContent>
   </xs:complexType>
   <xs:complexType name="shortNumberParameter">
      <xs:simpleContent>
         <xs:restriction base="parameter">
            <xs:pattern value="[0-9]{1,10}"/>
         </xs:restriction>
      </xs:simpleContent>
   </xs:complexType>
   <xs:complexType name="longNumberParameter">
      <xs:simpleContent>
         <xs:restriction base="parameter">
            <xs:pattern value="[0-9]{1,20}"/>
         </xs:restriction>
      </xs:simpleContent>
   </xs:complexType>
   <xs:complexType name="extendedShortNumberParameter">
      <xs:simpleContent>
         <xs:restriction base="parameter">
            <xs:pattern value="[-]?[0-9]{1,11}"/>
         </xs:restriction>
      </xs:simpleContent>
   </xs:complexType>
   <xs:complexType name="decimalParameter">
      <xs:simpleContent>
         <xs:restriction base="parameter">
            <xs:pattern value="[0-9]{1,3}\.[0-9]{1,1}"/>
         </xs:restriction>
      </xs:simpleContent>
   </xs:complexType>
```

```
  <xs:complexType name="doubleParameter">
    <xs:simpleContent>
      <xs:restriction base="parameter">
        <xs:pattern value="[0-9]{1,10}\.[0-9]{1,3}"/>
      </xs:restriction>
    </xs:simpleContent>
  </xs:complexType>
  <!-- Main definition -->
  <xs:complexType name="openSFDefinition">
    <xs:sequence>
      <xs:element name="descriptors" type="descriptorSet" minOccurs="0"/>
      <xs:element name="ios" type="ioSet" minOccurs="0"/>
      <xs:element name="tools" type="toolSet" minOccurs="0"/>
      <xs:element name="stages" type="stageSet" minOccurs="0"/>
      <xs:element name="models" type="modelSet" minOccurs="0"/>
      <xs:element name="simulations" type="simulationSet" minOccurs="0"/>
      <xs:element name="modelofsims" type="modelsOfSimulationSet"
minOccurs="0"/>
      <xs:element name="sessions" type="sessionSet" minOccurs="0"/>
      <xs:element name="simofsessions" type="simulationsOfSessionSet"
minOccurs="0"/>
      <xs:element name="paramofsessions" type="paramOfSessionSet"
minOccurs="0"/>
      <xs:element name="ioofsessions" type="iosOfSessionSet" minOccurs="0"/>
      <xs:element name="toolofsessions" type="toolsOfSessionSet"
minOccurs="0"/>
      <xs:element name="timeofsessions" type="timesOfSessionSet"
minOccurs="0"/>
      <xs:element name="modelperturbations" type="modelPerturbationSet"
minOccurs="0"/>
    </xs:sequence>
    <!-- Foreign keys -->
    <xs:assert test="every $io in ios/io satisfies (some $other in
descriptors/descriptor satisfies (data($other/field[@name = 'id']) eq
data($io/field[@name = 'descriptorId'])))" />
    <xs:assert test="every $param in paramofsessions/paramofsession satisfies
(some $other in sessions/session satisfies (data($other/field[@name = 'id']) eq
data($param/field[@name = 'sessionId'])))" />
    <xs:assert test="every $io in ioofsessions/ioofsession satisfies (some
$other in sessions/session satisfies (data($other/field[@name = 'id']) eq
data($io/field[@name = 'sessionId'])))" />
    <xs:assert test="every $sim in simofsessions/simofsession satisfies (some
$other in sessions/session satisfies (data($other/field[@name = 'id']) eq
data($sim/field[@name = 'sessionId'])))" />
    <xs:assert test="every $sim in simofsessions/simofsession satisfies (some
$other in simulations/simulation satisfies (data($other/field[@name = 'id']) eq
data($sim/field[@name = 'simId'])))" />
    <xs:assert test="every $model in modelofsims/modelofsim satisfies (some
$other in models/model satisfies ((data($other/field[@name = 'id']) eq
data($model/field[@name = 'modelId'])) and (data($other/field[@name =
'version']) eq data($model/field[@name = 'modelVersion']))))" />
    <xs:assert test="every $model in modelofsims/modelofsim satisfies (some
$other in simulations/simulation satisfies (data($other/field[@name = 'id']) eq
data($model/field[@name = 'simId'])))" />
    <xs:assert test="every $time in timeofsessions/timeofsession satisfies
(some $other in models/model satisfies ((data($other/field[@name = 'id']) eq
data($time/field[@name = 'modelid'])) and (data($other/field[@name = 'version'])
eq data($time/field[@name = 'modelVersion']))))" />
     <xs:assert test="every $time in timeofsessions/timeofsession satisfies
(some $other in sessions/session satisfies (data($other/field[@name = 'id']) eq
data($time/field[@name = 'sessionid']))))" />
     <xs:assert test="every $tool in toolofsessions/toolofsession satisfies
(some $other in sessions/session satisfies (data($other/field[@name = 'id']) eq
data($tool/field[@name = 'sessionId']))))" />
   </xs:complexType>
</xs:schema>
```

**END OF DOCUMENT**