estec

European Space
Research
and Technology Centre
Keplerlaan 1
2201 AZ Noordwijk
The Netherlands
T +31 (0)71 565 6565
F +31 (0)71 565 6040
www.esa.int

DOCUMENT

ESA generic E2E simulator Interface Control Document

ESA generic E2E simulator Interface Control Document 1.2.5 final.docx

Prepared by	**DME and ESA OpenSF Team**
Reference	**PE-ID-ESA-GS-464**
Issue	**1**
Revision	**2.5**
Date of Issue	**10-07-2019**
Status	**Release**
Document Type	**ICD**
Distribution	**all**

European Space Agency
Agence spatiale européenne

APPROVAL

Title ESA generic E2E simulator Interface Control Document	
Issue 1	**Revision** 2.5
Author DME and ESA openSF team	**Date** 10-07-2019
Approved by	**Date** 10-07-2019
Michele Zundo	

CHANGE LOG

Reason for change	Issue	Revision	Date
First Issue	1	0	12-05-2015
Added Time Based scenario Orchestration	1	1	20-08-2015
General clean-up and removal of obsolete feature in line with openSF 2016	1	2	13-04-2016
Updates of Time orchestration file format to update obsolete examples and coorect discrepancy	1	2.1	15-08-2016
Added definition of elementType and editorials	1	2.2	15-10-2016
Corrected typos in type range definition and ref. doc update	1	2.3	21-08-2017
Updates addessing issues found during development of openSF 3.7.2	1	2.4	30-05-2018
Use a coherent syntax for modes definition between Timeline scenario and Local Configuration file, clarification on parameters' format and on path usage	1	2.5	10-07-2019

CHANGE RECORD

Issue 1	**Revision** 0		
Reason for change	**Date**	**Pages**	**Paragraph(s)**
First Issue	12-05-2015	all	All

Issue 1	**Revision** 1		
Reason for change	**Date**	**Pages**	**Paragraph(s)**
Added Time Based scenario Orchestration	20-08-2015	22-26	§3.5

Issue 2	**Revision** 2		
Reason for change	**Date**	**Pages**	**Paragraph(s)**
General cleanup of document structure	13-04-2016		all
Added definition of matrix and array parameter types	13-04-2016		Section 2

Removal of legacy syntax for conf file (old openSF)	13-04-2016		Section 2
Consolidation of guidelines and example in section 4	13-04-2016		Section 3

Issue 2	**Revision** 2.1		
Reason for change	**Date**	**Pages**	**Paragraph(s)**
Updated version of reference/applicable documents	15-08-2016	Section 1.6, 1.7	all
Clarified naming convention	15-08-2016	Section 2.2.3	all
Removed obsolete attributes	15-08-2016	Section 2.2.6.1	all
Corrected XML examples	15-08-2016	Page 19, 21,	all
Harmonised naming Tag and Parameters' and removed redundant parameters	15-08-2016	Section 2.3.2, 2.3.3	all

Issue 2	**Revision** 2.2		
Reason for change	**Date**	**Pages**	**Paragraph(s)**
Added definition of elementType	15-10-2016	19	2.2.6.2
Editorials (upper case/lower case)	15-10-2016	19	2.2.6.2

Issue 2	**Revision** 2.3		
Reason for change	**Date**	**Pages**	**Paragraph(s)**
Updated reference documents	21-08-2017	9	1.6
Corrected typos in INT/FLOAT range definition	21-08-2017	19	2.2.6.2

Issue 2	**Revision** 2.4		
Reason for change	**Date**	**Pages**	**Paragraph(s)**
Corrected timestamp format according to CCSDS	30-05-2018	17	2.2.4
Defined type for TIME parameter compatible with CCSDS ASCII	30-05-2018	19	2.2.6.2
Clarified description of structured and simple types	30-05-2018	19-20	2.2.6.2
Removed schema spec	30-05-2018	22	2.2.6.2
Corrected typo in examples from DOUBLE to FLOAT	30-05-2018	many	

Issue 2	Revision 2.5		
Reason for change	**Date**	**Pages**	**Paragraph(s)**
Clarified usage of path and parameter format	10-07-2019		2.1.2, 2.2.6.2
Added enclosing <ModuleExecutionModes> tag to local configuration file syntax	10-07-2019		2.3.2
Clarify text regarding Local Configuration file wrt modes	10-07-2019		2.3.2
Remove redundant <ModuleExecutionModes> from Timeline file	10-07-2019		2.3.2

Table of contents:

1 INTRODUCTION

1.1 Purpose

An E2E Performance Simulator consists of a set of software modules simulating the space segment, its data output and the subsequent ground retrieval (level 1 and Level 2). The execution of these software modules needs to be orchestrated including in particular invocation and provision of input data. The definition of a set of standardised conventions and requirements, which the modules have to adhere to, allows then the use of a common orchestrating framework.

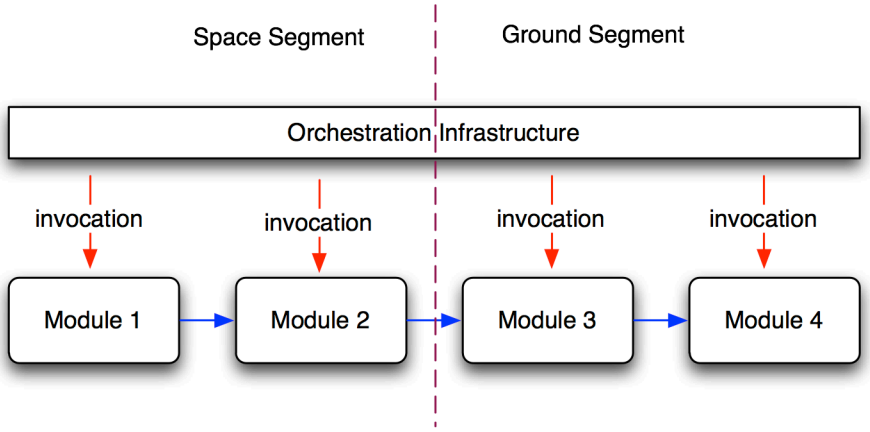

Figure 1-1: E2E Performance Simulator

These general conventions have been built based on the experience in E2E development for a number of different Earth Observation missions as well as from the experience gathered in the development and evolution of the current standard ESA E2E orchestrating framework OpenSF.

This document is describes therefore a common, generic interface for software modules, which also allows their integration into the ESA E2E Performance Simulator orchestrating infrastructure. These software modules can consist (but not only) of scene generators, instrument or platform simulators and processors or analysis tools. This interface has been designed to be lightweight and can be easily added to existing modules and it is compatible with many of the existing E2E Performance Simulator developed during Phase 0/A in Earth Observation as well with the orchestrating infrastructure [OPENSF] currently used within ESA.

This document describes as well the best practices to be followed by coding a module participating into an E2E Performance Simulator as well as the way modules are used and operated within an orchestrating infrastructure (e.g. OpenSF).

While each developer can implement the ICD by itself, to facilitate the development ESA makes available a set of reference libraries that already implement the interface according to this document [OSFI].

While ICD is generic, it notes where relevant the ***legacy*** aspects originating by the use of modules within the [OPENSF], e.g. when some feature is not yet supported in [OPENSF] or when some construct is allowed for compatibility.

1.2 Scope

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 details the orchestrating framework interfaces and gives some development guidelines.

Potential readers of this document include scientists/engineers and modellers interested in integrating their development into ESA E2E Simulator Orchestrating Framework.

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	Applicable Document
API	Application Programming Interface
COTS	Commercial Off-The-Shelf
CWD	Current Working Directory
E2E	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
RD	Reference Document
TBC	To Be Confirmed
TBD	To Be Defined

1.4 Definitions

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

Table 1-2: Relevant definitions table

Definition	Meaning
Module	Executable entity that can take part in a simulation. A module can be understood, broadly speaking, also as an "algorithm". Basically, it contains the recipe to produce products function of inputs. A module contains also several rules to define the input, output and associated formats. Furthermore, its behaviour is controlled by two configuration files. Overall, the architecture of a module consists of: ➢ The source code and its binary compiled counterpart ➢ A global configuration file with parameters common to several modules ➢ A local configuration file with module specific parameters ➢ An input file that characterizes its inputs ➢ An output file that characterizes its outputs Modules are not considered part of the framework.
Simulation	A simulation is understood as a list of modules (or even a module alone) that is run sequentially and produces observable results.
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.[1]
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.
Configuration File	An XML file that contains all the parameters necessary to execute a module. A configuration file instance must comply with the corresponding XML schema defined at module creation time.
Parameter	A constant whose value characterizes a given particularity of a module. Parameters are user-configurable, they are fixed before launching a module and, for practical reasons, and not all of them shall be accessible from the HMI.
Batch mode	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.

[1] This definition is planned for removal in future version of this ICD to be substituted by "Execution"

1.5 Applicable Documents

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

Table 1-3: Applicable Documents

Reference	Code	Title

1.6 Reference Documents

The following table specifies the reference documents that, while not binding, provide additional information.

Table 1-4: Reference documents

Reference	Code	Title	Issue
[OPENSF]	openSF-DMS-SUM-001	OpenSF User Manual Document	3.14
[OSFI]	openSF-DMS-OSFI-DM-013	OpenSF Integration Libraries Developers Manual	1.16
[CFI_FS]	PE-ID-ESA-GS-584-1.1	EO Mission SW File Format Specification	1.4
[EO-CFI]	(http://eop-cfi.esa.int/index.php/mission-cfi-software/eocfi-software)	Earth Observation Mission SW CFI	4.17

1.7 Standards

The following table specifies the standards used in this ICD.

Table 1-5: Standard

Reference	Code	Title	Issue
[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
[T_CCSDS]	CCSDS 301.0-B-4	CCSDS Blue Book,Time Code Formats	B-4,Nov 2010
[EO-FFS]	PE-TN-ESA-GS-0001	Earth Observation File Format Standard	3.0

2 INTERFACE DEFINITION OF E2E MODULES

An E2E Simulator requires a series of software modules to be executed in order with the output of some models used as input to others. To implement these operations an E2E orchestrating infrastructure in charge of invoking the software modules and passing them the appropriate inputs is needed.

This section defines the generic E2E Simulator Interface by giving a description of how to integrate a module (e.g. simulators, processors, tool, etc.) into a compliant E2E orchestrating framework and providing in particular harmonised requirements in 3 areas:

- Data I/O interface: read input files and write output files,
- Logging and Error Handling interface : provide common log/error messaging, and
- XML Configuration interface: use the XML files configuration interface,

A module is defined as an entity represented by a single executable program or script. These executables represent the smallest component within a simulation chain, and can perform a given scientific algorithm, instrument modelling, data processing or any desired part of the processing chain to be simulated.
In order to integrate modules into an orchestrating framework, developers shall use an interface convention as follows:

- A common calling format from the command line (shell, cmd), as per section 2.1.2.
- Logging messages format for user information and error handling described in sections 2.2.4 and 2.2.5.
- XML configuration files for user-configurable parameters described in section 2.2.6.

Below these lines Figure 2-1 shows the diagram of the module interfaces.

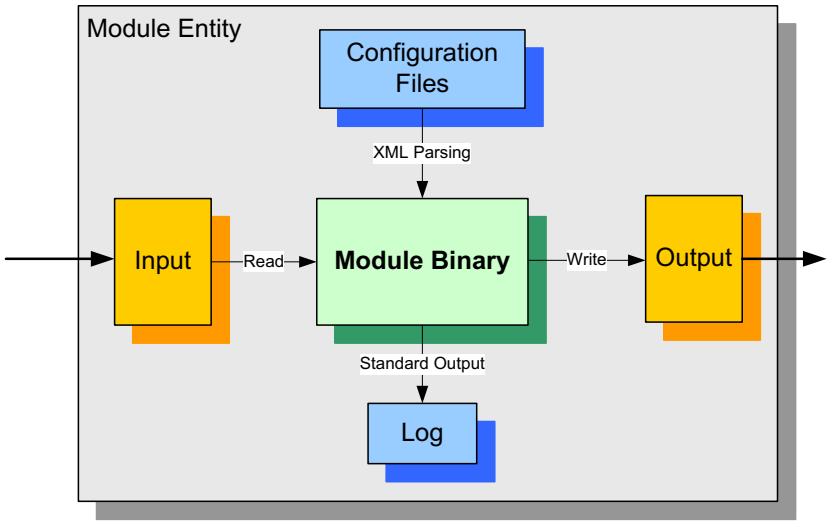

Figure 2-1: Module Entity Interfaces

A compliant module has to adhere to the following:

- **Environment variables** – modules can make use of environment variables passed from the platform to the execution environment.
- **Command line arguments** – module executables must accept a defined list of arguments.

- **Logging** – module output messages must comply with a given format.
- **Error handling** – successful executions must return a zero code to the operating system.
- **Configuration files** – modules can accept a specific XML format file if they want users to access and control their parameter values.

For a better understanding of modules logic, Figure 2-2 shows the flow diagram that a module shall nominally follow. The module execution flow can also be as complex as module developer wants provided that the interface compatibility is ensured on the item mentioned above.

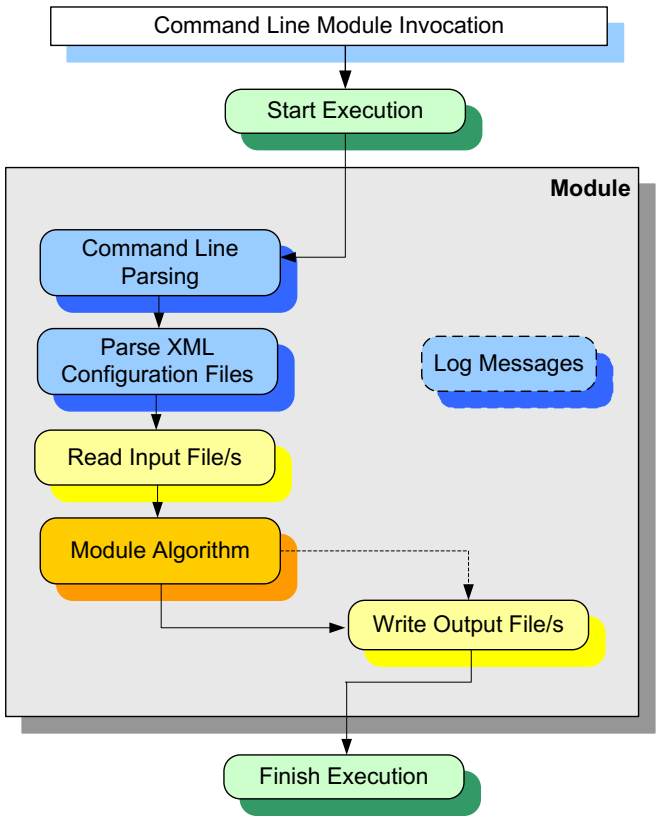

Figure 2-2: Normal Module Execution Flow Diagram

2.1 Module Execution

2.1.1 Environment variables

While users and modules can define a set of environment variables before the simulation execution that will be active during the whole execution process and used by either the orchestrating infrastructure or by the

modules, one particular environment variable is set by the orchestrating framework: *E2E_HOME*[2]. The *E2E_HOME* environment variable is defined and stores the location of the orchestrating framework software; the modules can therefore assume it is defined and use it accordingly.[3]

In addition, for each simulation session, a *session folder* is created by the E2E orchestrating framework at *$E2E_HOME/sessions/<unique_session_id>* and a corresponding *E2E_SESSION_HOME* environment variable is defined storing the location of the session folder. This environmental variable can be used inside the configuration file to specify to the module a location within the session folder (e.g. to point to a common /log folder for all modules inside the session folder)

Alternatively, users can also use or define modules that do not use the *E2E_HOME* environment variable.

2.1.2 Command line arguments

The convention for invoking a module is via its command line arguments. The general format of the calling command is a list of tokens grouped as:

- Two configuration files: one Global and one Local
- Input files
- Output files

Command line shall adhere to the following format (described here in Extended Backus-Naur form [BNF]):

```
<command_line> ::= <executable_name> <whitespaces> <configuration_files> <whitespaces>
<input_files> <whitespaces> <output_files> <EOL>
<executable_name> ::= <file_name>
<whitespace> ::= (“ “)
<whitespaces> ::= <whitespace>+
<file_name> ::= (<alphanumeric>)+
<configuration_files> ::= <list_of_files>
<input_files> ::= <list_of_files>
<output_files> ::== <list_of_files>
<list_of_files> ::= <file_name>(“,” <file_name>)*
```

Examples of command line are:

```
> executable config_file_global.xml,config_file_local.xml input_file1 output_file_1
> executable config_file_global.xml,config_file_local.xml input_file_1,input_file_2
output_file_1,output_file_2
```

The first argument is the name of the binary (or executable shell-script).

---

[2] The openSF framework up to version 3.7.3 uses for this purpose the equivalent OPENSF_HOME environmental variable.

[3] The openSF framework uses the OPENSF_HOME environmental variable to expand the relative paths given by the user via the GUI to the local and global configuration file into absolute ones, retrieve the two files and copy them into the *session_folder* directory, then passes the new full absolute path of both to each module upon invocation.

The second argument is the global configuration file and followed/comma-separated with the module local configuration file. These files are XML format files that shall be provided with a related schema. The syntax of a configuration file is specified in section 2.2.6.

The next group of command-line argument is a comma separated list of input data file names (which may be of various different formats) while the last group of command-line arguments is a comma separated list of output data files (which also may be of various different formats).

Files declared in the command line arguments will be managed by an orchestrating framework as taking part of the simulation chain and to be used by other modules and therefore stored in the dedicated *session folder* during the simulation execution.

Note that file names and paths shall not include blank spaces and must be separated by commas only.
For the configuration, input and output file names can be either a full-path name or a relative path name.

Relative path name shall be interpreted by the module as relative to CWD.[4]

2.2 Module Interface

2.2.1 Input/Output Files

There are no constraints imposed on modules for reading input files or writing output files. Modules have the freedom to specify as input/output either a directory or a set of one or more files. In case a directory name is used, it is responsibility of each module to select the relevant/correct inputs from the directory. A compliant orchestration infrastructure will automatically trigger a module execution when all the input files are detected as present and the generating process has completed successfully.

2.2.2 Auxiliary Files

A compliant E2E orchestrating framework makes use of one separate directory per each session execution where it will copy the required input files not generated by the modules (e.g. input for the first module or initial data) . This copy includes directories and their content whenever the input is specified as such. To avoid disk space overuse the developer can define large input files as "auxiliary files".

An "auxiliary file" is therefore a data file that is used as input to the simulation/processing chain but that will not be duplicated and stored in the session execution folder e.g. of candidates are a very large orbit files, Databases, LUTs or reference data files which do not change (e.g. physical constants). The "auxiliary files" are consequently not declared in the command line as inputs but passed to the module as a parameter within the configuration file described in section 2.2.6) and therefore not processed/visible by the orchestrating framework[6].

NB Declaring input data as auxiliary files should be carefully assessed as it makes the data set within the execution directory incomplete and does not ensure that re-execution will be identical, as external auxiliary

[4] The openSF framework uses the name and path specified by user in the GUI for input files also to check availability before start assuming they are prepended by $OPENSF_HOME and then generates the absolute path when calling the module.
[6] When data is passed to a module as auxiliary file specifying a path in configuration files, this ICD does not require a method/syntax to allow the orchestrating infrastructure to check their existence at time a module is started.

data could have been changed and reduce the robustness of the E2E simulation chain as the existence of auxiliary files is not ensured by the orchestrating framework

2.2.3 Module interface file naming conventions

In order to:

a) facilitate the integration activities into a processing chain of several modules developed separately using coherent naming
b) allow the orchestration infrastructure to identify interfaces and automatically detect input/output data availability

each module is (logically) identified by a user defined *ModuleID* and shall use the naming conventions described here below.

- Global Configuration files shall be named `Global_Configuration.xml`;
- Local Configuration files shall be named with each (user defined) module identifier ModuleID followed by suffix "_Local_Configuration.xml" (e.g. `Geo_Local_Configuration.xml`);
- Timeline file shall be named with string "Timeline_" followed by a user defined variable part, e.g. `Timeline_Commissioning_20150712.xml`

Input/Output Simplified convention:

- Input directories or files shall be named with the module identifier followed by suffix "_Input" (e.g. `Iono_Input`);
- Output directories or files shall be named with the module identifier followed by suffix "_Output" (e.g. `Scene_Output`).

Input/Output Advanced convention: [7]

This convention supports the use and generation by modules of input/output files with filename having a fixed part and a variable part (e.g. the timestamp or sensing time). Its use is envisaged to support the generation/ingestion of realistically named data products.

- Names for Input/Output directories or files shall be identified by a fixed string and a variable part (regular expression).

[7]Responsibility to generate and use the filenames according to the advanced convention resides with the module developers .

2.2.4 Logging

The E2E orchestrating framework offers logging services based on the capture of the messages that each module triggers/launches. The requirement here is that the modules must send the messages to a standard output for that language (e.g. default unit number 6 in FORTRAN codes, *stdout* in C codes) following the next formatting described here below in Backus-Naur Form [BNF]:

```
<message> ::= (<progress> | <log>) <EOL>
<progress> ::= “Progress” <whitespace> <delimiter> <whitespace> <progress_body>
<delimiter> ::= “|”
<progress_body> ::= <integer> “ of “ <integer>
<log> ::= <type> <whitespaces> <delimiter> <whitespaces> <text> [<whitespaces>
<delimiter> <whitespaces> <version>]
<type> ::= “Error” | “Warning” | “Info” | “Debug”
<version> ::= <digit>(“.” <digit>)*
<digit> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
<whitespace> ::= (“ “)
<whitespaces> ::= <whitespace>+
```

This format defines five different types of messages:

- **Information**. This is an informative message raised by the module describing a harmless event. Module execution should continue with no interruptions.
- **Warning**. The module has detected a non-fatal error or anomalous condition in data or during the processing that may cause a fatal error or affect the outputs in format or content. The execution should continue with no interruption.
- **Error**. A major error has happened in the module execution, the module has detected it and has time to “graciously” close the execution or handle it in a module-specific way.
- **Debug**. Detailed information of the module execution given to the user. Information is intended to lead the user (or module developer) to support fixing a problem. This is a harmless event so module execution should continue with no interruptions.
  Modules shall present debug messages only if the environment variables named “DEBUG_MODE” is defined and set as “On” in the module execution context.
- **Progress**. Numerical information on the amount of module execution performed.

Examples of log messages provided by an executable are:

```
Info | Method “m” started | 3.2
Warning | Method “m” applied an approximation to this calculation
Error | Method “m” could not converge to a solution
Debug | Method “m” obtaining error = 0.001
Progress | 5 of 25
```

It is important to recall that it is the responsibility of the developer to decide when a condition shall be flagged as a *Warning* instead of an *Error*, or when and where a *Debug* message is relevant. Therefore, it is not ensured that all “*Warning*” messages shown in the orchestrating framework have the same severity and consequences, i.e. they are qualitatively the same among the different modules.

All those pieces of information composing a message must be joined together by the module that has to write the complete message as string of characters formatted according to this ICD. The version component of the message is optional and represents the version of the logging library used to issue the message.
If the message does not fulfil this requirement it will not be considered as information to be stored, thus, when closing a session, it will be lost.

The time-stamping of each message with actual computer time is the responsibility of the logging infrastructure implementing this ICD; any other time information relative to the domain of the processing module (e.g. sensing time associated with a certain data causing the message or scenario simulation time) has to be added as part of the message. Any such time representation is recommended to abide to the ASCII format as described in section 2.5 of [T_CCSDS] with millisecond decimal part and without the optional Z terminator 'YYYY-MM-DDThh:mm:ss.nnn', e.g. `2015-06-04T12:53:48.021`

2.2.5 Error handling

An orchestrating framework implementing this ICD will interrupt the execution of a simulation once a module returns a non-zero code. This is used to detect module crashes and can be adopted by module developers as another way to stop the simulation execution in case of errors. There is a basic difference between the return codes and the *Error* string message:

- When a non-zero code is detected, it means that an error condition has happened and that the module has been unable to detect it, handle it or is intentionally returning a non-zero code to signal to the E2E orchestration simulator that the whole session has to be aborted.
- When a string message of category "Error" is returned, it means that the module has found a non-nominal situation according to the flow and tests coded by the module developer. Thus, after an *Error* message, all the subsequent actions will proceed with what the module handling strategy foresees with no intervention by the E2E orchestration framework. While one option is to stop the module it can also take corrective actions so that the module proceeds with further processing. These possibilities are fixed by the developer at the time of coding it.

2.2.6 XML configuration files

The behaviour of a given module can be controlled using the two XML configuration files as previously described. This section describes the rules and conventions that such configuration files have to follow in addition to standard [XML] rules, e.g. the preamble `<?xml version="1.0" encoding="UTF-8"?>`.

Note that, the choice has been made to limit the parameter representation in the configuration files to a fix set of types in order to reduce complications.

The configuration files shall contain the following xml elements:

2.2.6.1 Element "<group_name>"

This type of element constitutes an informative name related to the scope of the elements it encloses. In addition to the mandatory cases listed below, the string *group_name* can be any user defined text.
An element of type <group_name> can contain nested groups that can in turn enclose parameters. When a <group_name> tag encloses a set of parameters, it is used by the orchestrating infrastructure as a label to identify them.

A mandatory <group_name> tag identifying the configuration logical file name shall be used in each file as listed below. This element constitutes the root xml element and encloses the whole file and it is a string identifying the configuration file logical name. For a module called "*ModuleID*" the following logical names shall be used for the each of the 3 configuration files identified in this ICD:

- `<Global_Configuration version="XX.YY.ZZ">`, (for Global Configuration File)
- `<`*`ModuleID`*`_Local_Configuration version="XX.YY.ZZ">` (for Local Configuration Files)
- `<Timeline_`*`UserDefinedString`* `version="XX.YY.ZZ">` (for Timeline Files as described in section 2.3.2)

NB Even if the configuration filenames violate the file naming requirements described in section 2.2.3, the Configuration File Logical name shall be as specified above

2.2.6.2 Element <parameter>

Sequence of one or more tag with fixed string "parameter". This element can define the following attributes:

- **name**. This is the parameter identifier. Names cannot contain spaces;
- **description**. Short definition or meaning for the parameter;
- **type**. The data type of the parameter. Possible values are system-independent type of values, only intended for formatting validations. The data types supported by configuration files are the following:

  Simple types
  - STRING. A string of alphanumeric characters with a size not greater than 255.
  - INTEGER. Integer number (no decimal part) between -231 and 231-1
  - FLOAT(legacy note);[8]. Decimal number as per IEEE 754 between 5.0×10^{-324} and 1.7×10^{308} (positive or negative)
  - BOOLEAN. TRUE or FALSE.
  - TIME A string according to CCSDS ASCII time format with millisecond or microsecond precision e.g. 2020-11-21T13:45:12.123 or 2020-11-21T13:45:12.123456
  - FILE. Path to a file in the local file system (either absolute or implicitly relative to *E2E_HOME* environment variable).
  - FOLDER. Path to a folder in the local file system (either absolute or implicitly relative to *E2E_HOME* environment variable).

  Structured types
  - ARRAY. A generic array of elements (up to 3 dimensions), where each dimension may have distinct sizes.
  - MATRIX. A special case of ARRAY of two dimensions in which all rows (sub-elements) have the same size[9].

  elementType
  - used in complex types to specify the type of the actual data. Its value can be any of the simple types described above. Note that heterogeneous arrays (with data elements having different *elementType* values, like a row of INTEGERs and a row of STRINGs) are not supported

[8] The Java types "int" and "double" fix the range of values in INTEGER and FLOAT parameters
[9] The dims for a MATRIX variable are specified as "columns rows", instead of having dims="rows" in the outer element and dims="columns" in each row

- **value**. (**UNSUPPORTED**) This is the numerical, string or file location value of the parameter *(legacy note);*[10]
- **units**. Physical units of measurements if applicable. This attribute is optional;
- **min**. Numerical minimum value for the parameter. Only applicable for FLOAT and INTEGER types.
- **max**. Numerical maximum value for the parameter. Only applicable for FLOAT and INTEGER types.
- **dims**. Size of the dimensions. For example: "5" defines a [5x1] vector of five elements (arranged in a row); and "4 3" is a matrix of [4x3] elements.
  By convention, to describe a two dimensional type the first dimension refers to columns and the second to rows. In case of a tree dimensional type, the dimension referring to layers becomes the last, as in [column x row x layer].

For scalar variables, the "dims" attribute can be avoided.

To support structured parameters and matrices (or more generically arrays of elements) the following syntax is prescribed *(legacy note)*[11].

1. to represent a vector, arranged in a row (with dimension [5x1])

```
<MyModule_Local_Configuration version="01.03.00">
  <parameters>

    <parameter name="myV" description="example 1" type="ARRAY" elementType="INTEGER" dims="5">
      1 2 3 4 5
    </parameter>

  </parameters>
</MyModule_Local_Configuration>
```

2. to represent a matrix (with dimension [4x3]) (*see attribute note*)[12]

```
<MyModule_Local_Configuration version="01.03.00">
  <parameters>

    <parameter name="MyMatrix" description="example 2a" type="MATRIX" dims="4 3">
      <parameter name="v1" type="ARRAY" elementType="INTEGER">1 2 3 4</parameter>
      <parameter name="v2" type="ARRAY" elementType="INTEGER">5 6 7 8</parameter>
      <parameter name="v3" type="ARRAY" elementType="INTEGER">9 10 11 12</parameter>
    </parameter>

  </parameters>
</MyModule_Local_Configuration>
```

3. to represent a two-dimensional generic array (considering the variable number of columns, the enclosing dimension is [6x4])

---

[10] Specification of the parameter values via the Value attribute is a legacy deprecated syntax used up to OpenSF version 3.3 and not supported by the present version of the ICD and by newer version of openSF.
[11] The simplified format for vector/matrices (based on simple concatenation of types) defined in ICD version 1.1 and accepted by openSF up to version 3.4 is deprecated by the current version of the ICD and by newer version of openSF and not to be used.
[12] Considering the current implementation openSF/OSFI for structured elements, the ***name*** and ***description*** attributes can only be used to access the outermost <parameter> element. When associated to inner elements these attribute cannot be accessed with the existing openSF/OSFI API and are ignored.

```
<MyModule_Local_Configuration version="01.03.00">
  <parameters>
    <parameter name="spectrum" description="example 3a" type="ARRAY" dims="4">
      <parameter name="v1" type="ARRAY" elementType="INTEGER" dims="3">1 2 3</parameter>
      <parameter name="v2" type="ARRAY" elementType="INTEGER" dims="6">4 5 6 7 8 9</parameter>
      <parameter name="v3" type="ARRAY" elementType="INTEGER" dims="5">10 11 12 13 14</parameter>
      <parameter name="v4" type="ARRAY" elementType="INTEGER" dims="1">15</parameter>
    </parameter>
  </parameters>
</MyModule_Local_Configuration>
```

4. to represent a three-dimensional generic array (considering the variable number of columns, the enclosing dimension is [5x4x2])

```
<MyModule_Local_Configuration version="01.03.00">
  <parameters>
    <parameter name="OuterArray" description="example 3b" type="ARRAY" dims="2">
      <parameter name="innerArray1" type="ARRAY" dims="4">
        <parameter name="v1a" type="ARRAY" elementType="INTEGER" dims="1">1</parameter>
        <parameter name="v2b" type="ARRAY" elementType="INTEGER" dims="3">2 3 4</parameter>
        <parameter name="v3c" type="ARRAY" elementType="INTEGER" dims="2">5 6</parameter>
        <parameter name="v3d" type="ARRAY" elementType="INTEGER" dims="1">7</parameter>
      </parameter>

      <parameter name="innerArray2" type="ARRAY" dims="3">
        <parameter name="v1e" type="ARRAY" elementType="INTEGER" dims="2">1 2</parameter>
        <parameter name="v2f" type="ARRAY" elementType="INTEGER" dims="5">3 4 5 6 7</parameter>
        <parameter name="v3g" type="ARRAY" elementType="INTEGER" dims="2">8 9</parameter>
      </parameter>
    </parameter>
  </parameters>
</MyModule_Local_Configuration>
```

For vector/matrix types the values consists of blank-separated list of values by rows. To fully describe a parameter of a structured type, the attribute elementType is used to define the element type. Attribute *elementType* may have, as value, all currently supported parameter simple types.

String vectors must enclose each element in single quotes. For example `"'a string' 'second string' 'last string'"`. The situation for an array of strings can be handled with generality, as it is possible to manage strings that included blanks or commas. For example, for a case with `dims="1 3"` (an array of three strings) these are the possible options:

- Value set to 'PMT' 'PMT' 'PMT'. **OK.** 3 values
- Value set to P T P T P T **Invalid.** 6 values in a vector of size 3

Parameters of type FILE/FOLDER can include also environment variables explicitly, by enclosing their name in curly braces e.g. ${*E2E_HOME*}/var/tai-utc.data). A FILE/FOLDER parameter value expressed as a relative path is implicitly considered as being prefixed by "${E2E_HOME}/".

A compliant orchestrating framework or associated functions automatically resolve FILE/FOLDER parameters, by substituting explicit or implicit environment variables and resolving the path to an absolute path. When retrieving these type of parameters from the configuration file, a simulation module will therefore always access an absolute path.

2.2.6.3 Validation Schemas

An XSD schema can be used by the module to validate the module configuration files.

The freedom allowed in the use of group names in XML configuration files makes impossible the creation of a unique XSD schema file valid to all of them. Note that module developers are able to create as many groups as they desire and there are no restrictions in the number of nesting levels.

2.2.6.4 Adoption of Earth Observation File Format

The use of the Earth Observation File Format [EO-FFS] is oriented to ground segment software in a near-operational environment. This format is not used for the XML configuration files defined in this interface but its use is recommended as a standard, harmonised format for the input/output files and data used by each module. This is strongly recommended, in particular, for the elements that simulate the ground processing (e.g. Level 1/Level 2 Processor Prototypes).

Note that if any of the software modules makes used of Earth Observation Mission SW CFI libraries [EO-CFI] e.g. to support orbital propagation, visibility calculations and read/write of XML files, then these files are fully compliant with the [EO-FFS].

2.3 Designing E2E modules for Time Based orchestration

This section prescribes how to define the standard configuration interface of each module in order to allow invocation in a time based fashion from any complaint orchestrating framework.

In a real operational satellite system, the behaviour and stimuli presented to the satellite are a function of time and a dynamic function of the operational control by the ground or on-board. Depending on instrument mode, attitude, position different stimuli are available and different on-board functions active (e.g. different observation modes, calibration, downlink, attitude determination, etc). Being able to flexibly drive the modules of an E2E simulation from the infrastructure as if flying the actual mission allows an efficient execution of the simulation, the assessment of mission performance in representative operational scenario, a quick way for test data generation and in general gives the freedom to explore different operational concepts in an automated way and not restricted to feeding static test stimuli to the simulation modules and avoiding the need to hardcode dynamic behaviour inside static data.

The Time based scenario execution implements the notion of time driven execution of a simulation whereby each simulation module is invoked in a sequence of time segment. The implementation was designed to maintain interfaces compatibility with modules compliant with previous orchestrating infrastructures operating mode (e.g. openSF up to version 3.3).
The interface described in this ICD therefore (a) keeps the previous approach for invoking a module (as per section 2.1.2); (b) naturally extends existing configuration files (defined in section 2.2.6) to include additional time related parameters.

This section introduces the concepts and the definition in term of the interfaces also with regards to the reference [OPENSF] orchestrating infrastructure that implements this ICD.

2.3.1 Concepts

2.3.1.1 E2E chain execution categories

To guide the time driven execution each module in an E2E chain are classified in one of two execution categories: *Simulation* and *Processing*. A module of the "*Simulation*" category can be run either in *time-driven* execution or in *data-driven* execution, while a module of the "*Processing*" category is instead run only in data-driven execution.

2.3.1.2 Time-driven vs. Data-driven execution

Each overall E2E chain/session may be performed either in time-driven or data-driven execution *(legacy note)*[13]
When *in time-driven* execution the modules categorised as "Simulation" shall be executed in *time-driven* fashion (iterating in time) while in *data-driven* execution the same modules are executed in data-driven configuration (driven by data availability), on the other hand the modules categorised as "Processing" are always executed in data-driven configuration regardless of the overall session execution strategy therefore only for "Simulation"-type modules can time-driven configuration be applied.

[13]The "snapshot" execution strategy implemented in OpenSF up to version 3.2 where the whole processing chain is run without time reference - "static" mode - is a special case of the data-driven execution

2.3.1.3 Module-execution modes

Each module may have several associated module-execution modes as identified by its developer. Each module-execution mode is defined by a given set of parameters affecting the functions of that module, as example a parameter could be used to set the module to perform calibration simulation instead of measurements. For a given module with several available modes there may be parameters which are mode-independent (so applicable to all modes) and parameters, which are mode-specific. Nominally a mode is defined by a specific assignment of values to their corresponding mode parameters and these shall be passed by the orchestrating infrastructure to the module, it is however possible to manually override via the infrastructure these pre-defined values in a given execution although this is transparent to the module.

2.3.1.4 Timeline Segment

A timeline is composed by a time-ordered sequence of non-overlapping time segments, each defined by:

a. a start time;
b. one among: duration, number of steps and step size or end time;
c. a list of module-execution modes (one per each module of the E2E chain);
d. a status (active/inactive);
e. an (optional) set of overridden module execution mode parameters.

2.3.1.5 Session execution

In time-driven mode the orchestration framework invokes, for each timeline segment, the (simulation) modules in the order defined by the setup, each time passing as input the parameters of the selected mode defined by the timeline segment being executed. The process is repeated until the end of the timeline.

The time driven related parameters are passed to the modules thru the global configuration file grouped in an xml tag. The mode specific parameters are passed to each module thru the local configuration file grouped in an xml tag with the mode name. If the mode specific parameters were overridden in the timeline definitions this shall be signalled with an xml attribute in the mode group tag.

The orchestrating framework shall manage a single working directory for a session in time-driven execution (labelled with the execution timestamp). Within this working directory there shall be a separate working folder for each timeline segment (labelled with the simulation timestamp). A session in data-driven execution shall use a single working folder were the several executions of each given module shall access their inputs and outputs.

2.3.2 *Timeline Configuration*

The Timeline configuration is implemented via a file containing the information defining the time-driven execution of a session in a self-contained manner, i.e., containing all required parameterisation that allows the orchestrating infrastructure to execute and replicate a given time based simulation.

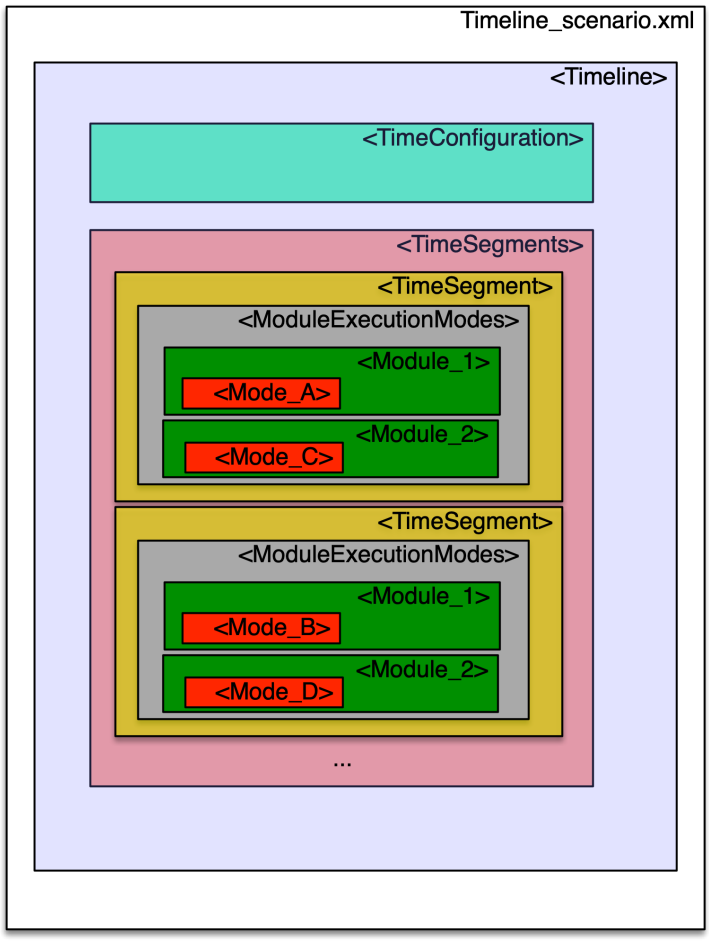

Figure 2-3 - E2E simulation with processing modules and simulation modules driven in time

The timeline configuration file is an XML file compliant with the configuration files format (defined in section 2.2.6) and with the logical structure shown in .

This file is not needed/present if the simulation is not time driven.

~~**Figure 2-3 - E2E simulation with processing modules and simulation modules driven in time**~~
The Timeline configuration file contains the following set of data:

a. the generic time parameters;

b. the list of time segments;

These 3 elements are described here below:

A. The generic time parameters are the following:
   - **InitialEpoch**. A Time indicated the start time of the entire time driven execution;
   - **DefaultTimeSegmentDuration**. An Integer with the default duration (in seconds) to apply when adding a new time segment to the timeline.

B. A time segment is defined by the following parameters:
   - **TimeSegmentStartTime**. A Time indicating the start time of the time segment;
   - **TimeSegmentDuration**. An Integer indicating the duration of the time segment (in seconds);
   - **Active**. An Boolean indicating whether the time segment is to be executed;
   - A set of **module execution modes**, identifying the mode in which each module shall execute during that time segment. It may optionally include a set of parameters re-defining module execution mode parameters used when the user intends to manually override the default values for a module execution mode.
     Each **module execution mode** is defined by the set of specific parameters grouped in an xml tag with the mode name.

The root element of the timeline configuration files shall be `<Timeline>.`

An example of a timeline configuration file is here below:

timeline_scenario.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Timeline version="00.15.33">
  <TimeConfiguration>
    <parameter name="InitialEpoch" description="Time Driven Execution Start Time"
               type="TIME">20150101T00:00:00.000</parameter>
    </TimeConfiguration>

    <TimeSegments>

      <TimeSegment>
        <parameter name="TimeSegmentStartTime" description="Time Segment Start Time"
                   type="TIME">20150101T00:00:00.000</parameter>
        <parameter name="TimeSegmentDuration" description="Time Segment Duration"
                   type="INTEGER" units="second">300</parameter>
        <parameter name="Active" description="Time Segment Status" type="BOOLEAN">TRUE</parameter>
        <ModuleExecutionModes>
          <GeometryModule>
            <Nominal></Nominal>
          </GeometryModule>
          <SceneGenerator>
            <Nominal status="override">
              <parameter name="toa_factor" type="FLOAT">1</parameter>
              <parameter name="pol_factor" type="FLOAT">0.02</parameter>
            </Nominal>
          </SceneGenerator>
        </ModuleExecutionModes>
      </TimeSegment>

      <TimeSegment>
        <parameter name="TimeSegmentStartTime" description="Time Segment Start Time"
                   type="TIME">20150101T00:05:00.000</parameter>
        <parameter name="TimeSegmentDuration" description="Time Segment Duration"
                   type="INTEGER" units="second">10</parameter>
        <parameter name="Active" description="Time Segment Status" type="BOOLEAN">FALSE</parameter>
        <ModuleExecutionModes>
          <GeometryModule>
```

```
        <Manouvre></Manouvre>
      </GeometryModule>
      <SceneGenerator>
        <Nominal></Nominal>
      </SceneGenerator>
    </ModuleExecutionModes>
  </TimeSegment>

</TimeSegments>

</Timeline>
```

2.3.3 *Module Configuration*

The parameters related to the time driven execution mode are passed to the modules thru the configuration file(s) grouped in the xml tag `<TimeExecution>`.

2.3.3.1 Global Configuration File

The fixed set of global time driven related parameters included in the global configuration file are the following:

- **InitialEpoch**. A Time indicated the start time of the entire time driven execution scenario.

global.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Global_Configuration version="01.00.00">

  <TimeConfiguration>

    <parameter name="InitialEpoch" description="Time Driven Execution Start Time"
               type="TIME">20150101T00:00:00.000</parameter>

    </TimeConfiguration>

</Global_Configuration>
```

Note that if time-driven execution is not implemented or supported by modules the corresponding time-related entries in the Global Configuration file might not be present.

2.3.3.2 Local Configuration File

The fixed set of parameters related to the module-specific time driven execution and that are included in each local configuration file are the following:

- **ModuleExecutionMode**. A String representing the module execution mode.
- **TimeSegmentStartTime**. A Time indicating the start time of the specific timeline execution step for the given module.
- **TimeSegmentDuration**. The seconds that define the duration of the time step to be executed by the module.

The mode specific parameters are passed to the module thru the existing local configuration file grouped in an xml tag with the mode name. If the module specific parameters were overridden in the timeline definitions this shall be signalled with an xml attribute in the mode group tag.

localA.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ModuleA_Local_Configuration version="04.15.33">

  <TimeConfiguration>
    <parameter name="ModuleExecutionMode" description="Module Execution Mode"
               type="STRING">Maintenance</parameter>
    <parameter name="TimeSegmentStartTime" description="Time Driven Execution Start Time"
               type="TIME">20150101T00:00:00.000</parameter>
    <parameter name="TimeSegmentDuration" description="Execution Length"
               type="INTEGER" units="second">300</parameter>
  </TimeConfiguration>

  <parameter name="Lambda" description="Central Wavelength"
             type="FLOAT" units="nm">500.3</parameter>
  <parameter name="ErrorThreshold" description="Threshold of the output RMS error"
             type="FLOAT" units="Km">1.0</parameter>

  <ModuleExecutionModes>
    <Off>
      <parameter name="calibration" description="text" type="BOOLEAN">TRUE</parameter>
    </Off>
    <Maintenance status="override">
      <parameter name="angle" description="text" type="FLOAT">0.5</parameter>
    </Maintenance>
  </ModuleExecutionModes>

</ModuleA_Local_Configuration>
```

Note that if time-driven execution is not implemented or the module is modeless, the corresponding time-related entries in the Local Configuration file will be not present.

3 GUIDELINES FOR DEVELOPMENT AND INTEGRATION OF E2E SIMULATORS

3.1 Coding guidelines and potential pitfalls

Developers should have in mind the following points when coding modules:

- Memory handling is responsibility of the module. This ICD does not require that a compliant orchestrating framework manages the memory assignments and destroys any data structure created by the module;
- A module can create child processes, but their management is responsibility of the parent module and not of the orchestrating framework;
- This ICD does not require that a compliant orchestrating framework detects when a module execution is "halted" or in an infinite loop unless a non-zero code is returned on exit. It is therefore recommended that logging information (see section 2.2.4) are issued around every two seconds to allows the orchestrating framework to update the progress bar informing, the user that there is no problem and to provide a more user friendly simulation environment.

3.2 E2E simulator development walkthrough

This section gives an example of the development process for a generic E2E simulation using modules compliant with this ICD.

The steps given in this section can be taken as guidelines for a correct integration of an E2E simulator. The simulation chain used along this section is depicted in Figure 3-1.

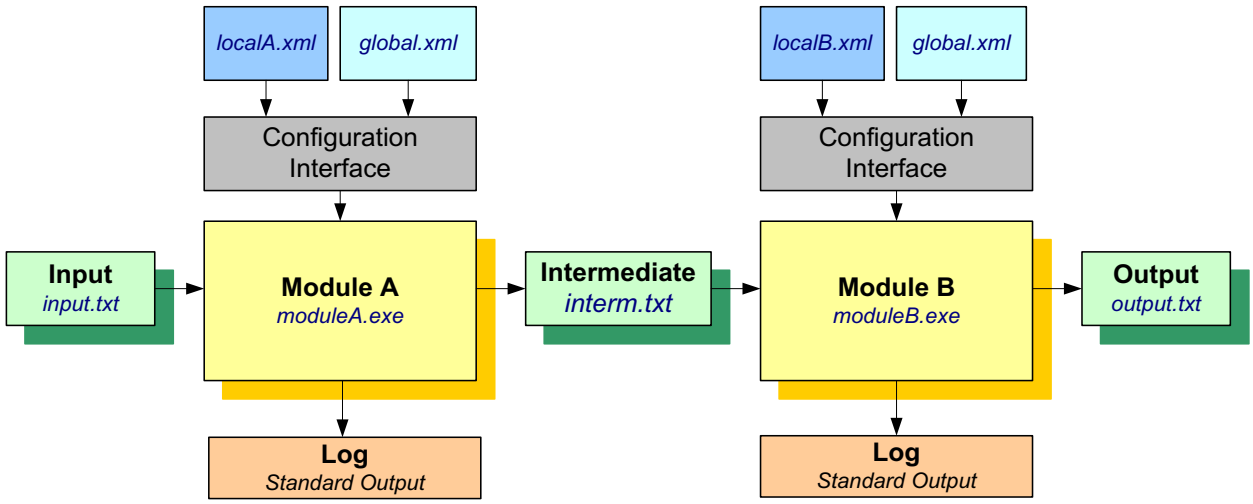

Figure 3-1: Fictitious Simulation Scenario

There are a series of steps that a developer has to address in order to accomplish a simulation goal and integrate all the system into a compliant orchestrating framework.

a) **Identify the elements involved in the simulation scenario.** In this example are:
   - Two modules – Module A and B (yellow boxes).
   - Three product files – Input, Intermediate and Output (green boxes).
   - Three XML configuration files – a global and two local files (blue boxes).

b) **Provide a detailed description of the simulation logic**. The one depicted in the example is as follows:
   - The input for the simulation chain is the file "input.txt", it is also the input for Module A.
   - There are two modules, one after the other, in the simulation (Module A and Module B). These modules execute a set of algorithms and are callable by command line invocation.
   - Module A generates an output file ("interm.txt") that Module B accepts as input.
   - The output of the simulation chain is the file generated "output.txt" and is considered as the result product in the chain.
   - Module A and Module B have a configuration interface requiring two files, one local ("localA.xml" and "localB.xml") and one global that is common to both modules ("global.xml"). These configuration files contain the simulation parameters that govern the internal functionality of modules. The ones relevant for all modules in the simulation shall be identified and assigned to global configuration file.
   - Identify which input data files are considered input files and which ones (if any) can be passed to the modules as path within the Local Configuration file noting the disadvantages mentioned in section 2.2.2
   - The log messages system provides information about the current state of the modules during a simulation run.

c) **Develop the modules following the architecture defined in the design phase.** The development process for the example shall be:

   - Build the configuration files with the simulation parameters. Simplified example configuration files are shown here below.

global.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Global_Configuration version="01.03.07">

  <parameter name="Nbands" description="Number of Bands" type="INTEGER">11</parameter>
  <parameter name="MissionID" description="Mission Identifier" type="STRING">Sentinel3</parameter>

</Global_Configuration>
```

localA.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ModuleA_Local_Configuration version="04.15.33">

  <parameter name="ErrorThreshold" description="Threshold of the output RMS error"
             type="FLOAT" units="Km">1.0</parameter>

</ModuleA_Local_Configuration>
```

localB.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ModuleB_Local_Configuration version="6.15.00">

  <parameter name="Lambda" description="Central Wavelength"
             type="FLOAT" units="nm">500.3</parameter>

</ModuleB_Local_Configuration>
```

- Implement the modules input/output interfaces. This step involves the reading of input files and configuration files (XML parsing), write the output files and the implementation of the command line calling interface as well as fetching the auxiliary files defined within the Local Configuration file..
- Implement the module logic, the algorithm.
- Perform Unit testing for the modules.

Once the development of the tasks above is completed, the test of manually running the simulation chain is possible the following commands:

```
> moduleA.exe global.xml,localA.xml input.txt interm.txt
> moduleB.exe global.xml,localB.xml interm.txt output.txt
```

d) **Define the simulation scenario within the orchestrating framework.**

e) **Verify the integration within the orchestrating framework by executing the simulation using the framework.**

3.3 Example Use Cases (time-driven)

Use Case 1: E2E simulation including both simulation modules and processing modules, when the simulation modules are driven in time-based synchronisation

From a conceptual point of view the expected behaviour of an execution in such scenario corresponds to what is depicted in Figure 3-2.

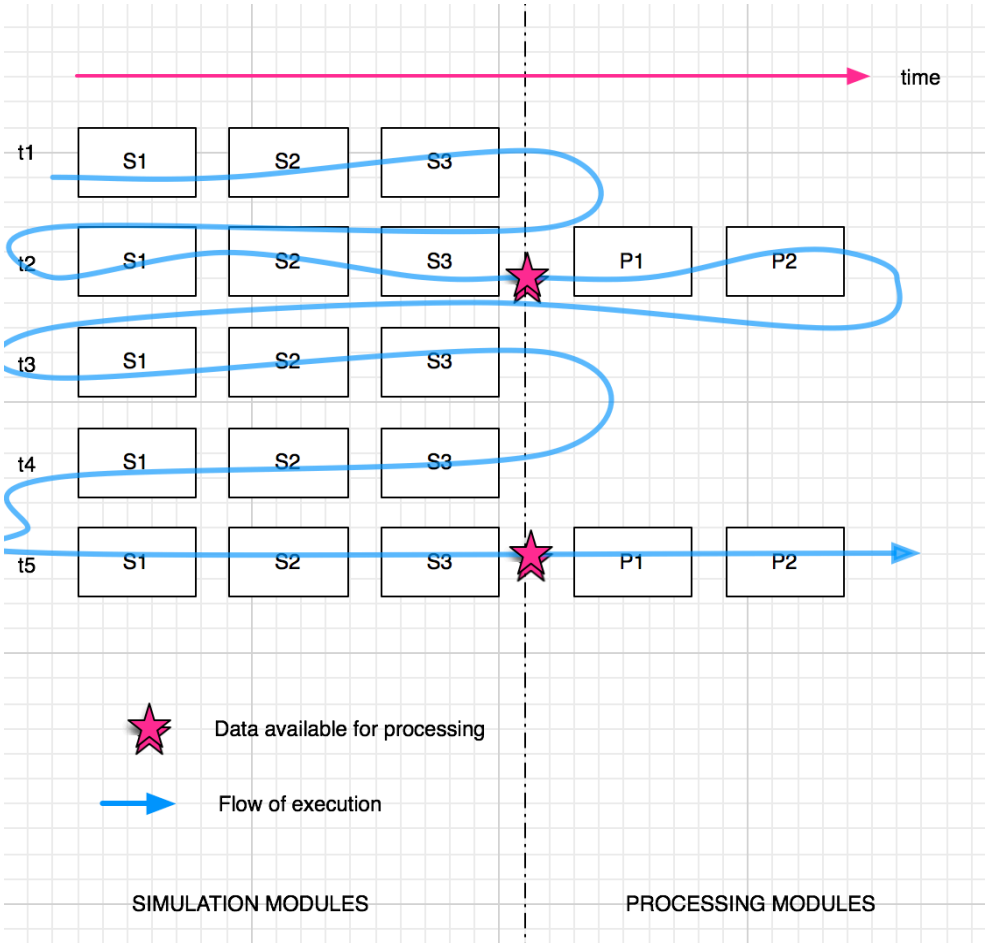

Figure 3-2 - E2E simulation with processing modules and simulation modules driven in time

Note that it is outside the orchestrating framework's orchestration scope to understand the events of "Data available for processing" (the "stars" in the figure above). These events depend on the processing chain definition as well as on the actual outputs of each module or even the particular implementation of the modules. Therefore the emulation of this flow can be performed in the following alternative ways:

a) always execute the entire processing chain (S1 -> P2) at each time moment, but P1+P2 shall only do "meaningful" computation if S3 has produced the consolidated outputs (identifiable by P1 through a given mask);
b) Execute the simulation in two separate steps:
    i. execute the time-driven execution, where the Simulation Module's part of the chain is executed, with one independent execution for each time moment;

ii. after concluding the above execution, trigger the data-driven execution, where the Processing Modules part of the chain are executed, configured to have as input the output of the previous time-driven execution;

Either way the underlying idea is that the responsibility for the "handover" between Simulation modules and Processing modules is either on the modules implementation or the operator, not on the orchestrating framework.

Use case 2: Multi-Instrument E2E simulation including both simulation modules (in time-driven execution) and processing modules, with different timelines for each instrument

In the example of Figure 3-3 consider that INSTR1 is in Nominal mode for 2 hours while INSTR2 changes every 10 second between CAL and NOM_A, NOM_B, NOM_C modes.

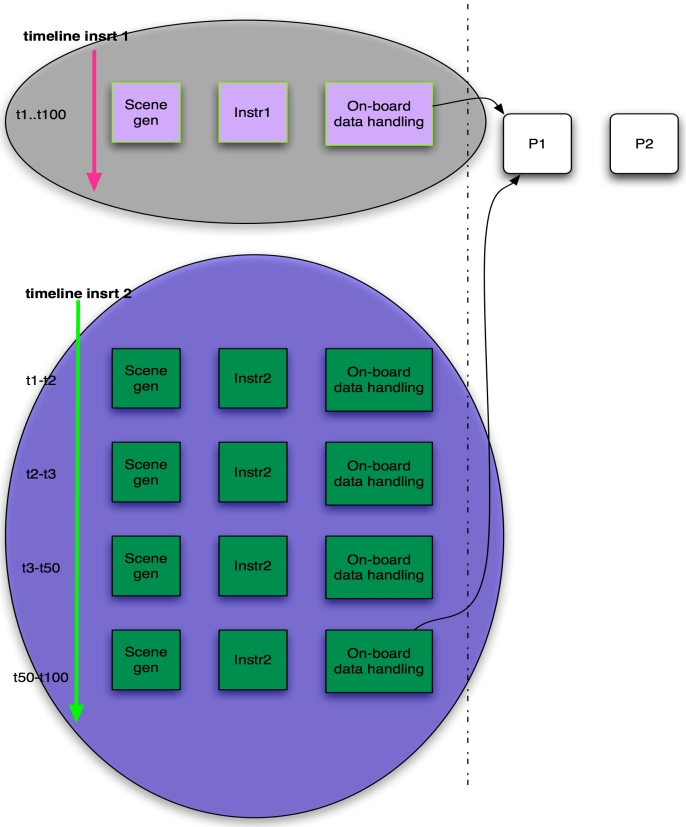

Figure 3-3 - Multi-Instrument E2E simulation with different timelines for each instrument

The shown E2E chain shall be achieved by having three separate simulations (one for Instr1, one for Instr2 and one for the "Processing" modules) executed separately and in sequence by the operator.

NB: this is independent of having or not the time-driven orchestration: a "static" version of the multi-instrument scenario depicted also requires separate simulation executions.