System Optimization: A Use Case in the Space Domain

Mihal Brumbulli*, Emmanuel Gaudin*, Alexandre Cortier**, Alain Rossignol**
*PragmaDev, Paris, France
**Airbus Defence & Space, Toulouse, France

***Abstract*—The JUpiter ICy moons Explorer mission, JUICE, is the first large-class mission in the Cosmic Vision 2015-2025 program of the European Space Agency. Planned for launch in 2022 and arrival at Jupiter in 2030, it will spend at least three years making detailed observations of the biggest planet in the Solar System and three of its largest moons, Ganymede, Callisto and Europa. JUICE will carry a total of 11 scientific experiments (instruments) to study the gas giant planet and its large ocean-bearing moons. Each instrument will do a number of readings producing streams of data, which will be stored in the on-board mass storage and then transmitted to Earth via a specific protocol. In this paper we present a model-based approach for system optimization in terms of instrument parameters, mass storage configuration, and transmission band allocation. We use standard modeling and testing languages with formal semantics for describing the system and the different configuration scenarios, which are then executed in co-simulation mode to produce (1) very precise results and (2) graphical comparison of different configurations to help trade off.**

***Index Terms*—System optimization, space mission, modeling and simulation, SDL, TTCN-3**

I. Introduction

The JUpiter ICy moons Explorer (JUICE) is the first large-class mission in the Cosmic Vision 2015-2025 program of the European Space Agency [1]. Planned for launch in 2022 and arrival at Jupiter in 2030, it will spend at least three years making detailed observations of the giant gaseous planet Jupiter and three of its largest moons, Ganymede, Callisto and Europa. The JUICE mission will address two themes of ESA's Cosmic Vision program: *What are the conditions for planet formation and emergence of life?* and *How does the Solar System work?*

The JUICE spacecraft will carry the most powerful remote sensing, geophysical, and in situ payload complement ever flown to the outer Solar System. The payload consists of 10 state-of-the-art instruments plus one experiment that uses the spacecraft telecommunication system with ground-based instruments. This payload is capable of addressing all of the mission's science goals, from in situ measurements of Jupiter's atmosphere and plasma environment, to remote observations of the surface and interior of the three icy moons.

All measurements in form of data streams coming from the 10 instruments will be stored in a Solid State Mass Memory (SSMM), and when telecommunication is possible they will be transmitted (downlinked) to ground. A successful and efficient operation of the system requires careful planing involving instrument configuration (e.g., frequency of measurements), mass memory organization (e.g., directories, files, etc.), and transmission band utilization. This paper presents a model-based approach to aid architects in finding the best configuration for an optimized operation of the system.

Section II defines the objectives of the use-case, while detailing the organization and operation of the system and listing the expected results. In section III we give an overview of the relevant technologies used in this model-based approach, and section IV describes the simulation model of JUICE. The simulation results are presented in section V, and some concluding remarks are given in section VII.

II. Objectives

A. Business Objectives

This use-case is part of an internal Airbus Defence & Space project to assess how model based techniques can improve the early maturity of the On-Board Functional Avionic requirements and design, in particular through the use of modeling tools allowing rapid prototyping and simulation at Functional Avionic Level. The objective is to make available to the Functional Avionics (FA) architects a set of Model Based Systems Engineering (MBSE) methods and tools which are adapted to the different operational situations FA architects are facing.

B. Project Objectives

The JUICE mission will carry out a total of 10 payload instruments, some of them producing a large amount of data. Each instrument is connected to the SSMM through a SpaceWire router to store output data flows as shown in figure 1.

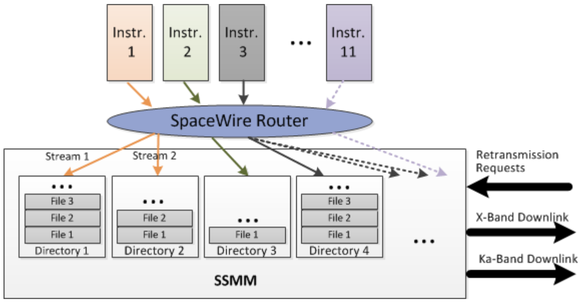


Fig. 1. JUICE use-case modeling overview.

On the downlink side, the deep space nature of the JUICE mission imposes a low transmission data rate and a large communication delay. To manage science data flow from instruments production to downlink, new operational concepts have been set-up:

- File Based Operation: Science data flows are stored as files in an SSMM and organized in directories. Creation of files is done according to switching criteria like maximum size or time.
- CCSDS File Delivery Protocol (CFDP): Automatic downlink of files according to directory's downlink priorities and retransmission of segments of files on ground requests in case of data corruption during downlink.

Each instrument produces data streams (up to 4 per instrument), possibly in parallel and the data rate of each instrument varies in time. Each data stream (up to 40) is connected to a directory. In this directory the data are stored in files. File switching (close the current file and open a new file to continue data recording) occurs upon time (since opening) or size criterion according to ground configuration. Files are transmitted to the ground in *X* or *Ka* band, during communication sessions (nominally 8 hours/day), with a priority management system. File selection for downlink is automatic, based on a priority that is a directory attribute. Data production can possibly continue during downlink and only closed files are downlinked.

C. Simulation Objectives

File and downlink management relies on several configurable parameters: the number of directories allocated to an instrument, the file switching criteria associated to each directory, the downlink priority allocated to a directory, etc. Data corruption may occur either during storage or during downlink (especially in *Ka* band). In this later case, the ground asks for data segment retransmission (files are deleted onboard only when correct/full reception is confirmed).

Due to these operational constraints (small bandwidth, error rate, downlink autonomy, and retransmission) and the configuration of the downlink management system, it is quite complex for an operation architect to anticipate the influence of these configuration parameters and thus to handle a concrete downlink scenario. In this context, a model capturing the science data flow from instrument production to downlink can help the architect and the Principal Investigators responsible of an instrument by providing simulation capabilities. The simulation model presented in this paper has been developed to help answering the following question: *How to operate the satellite to maximize the amount of transmitted data with respect to the scientific requests?* More precisely, the objectives of the simulation were to:

- assess data latency on-board (time between file closure and reception of this file on ground),
- assess the impact of downlink error and segment retransmission,
- help Principal Investigators in understanding the system and designing the data storage (file size, number of files, criteria for file switching, selective downlink usage), and
- assess memory utilization.

III. Technology

A. SDL

The Specification and Description Language (SDL) is a specification language defined by the International Telecommunication Union (ITU-T) in the Z.100 series [2]. SDL is targeted at the unambiguous specification and description of the behavior of reactive and distributed systems.

In SDL the overall design is called the *system*, and everything outside of it is defined as the *environment*. The system can be composed of *agents* and *communication constructs*. There are two kinds of agents: *blocks* and *processes*. Blocks can be composed of other agents and communication constructs, and they fulfill their functionality via processes. A process provides this functionality via extended finite state machines, and has an implicit queue for messages (or *signals*). Messages go through *channels* and end up in the processes queues. These concepts are illustrated in a simple example in figure 2.

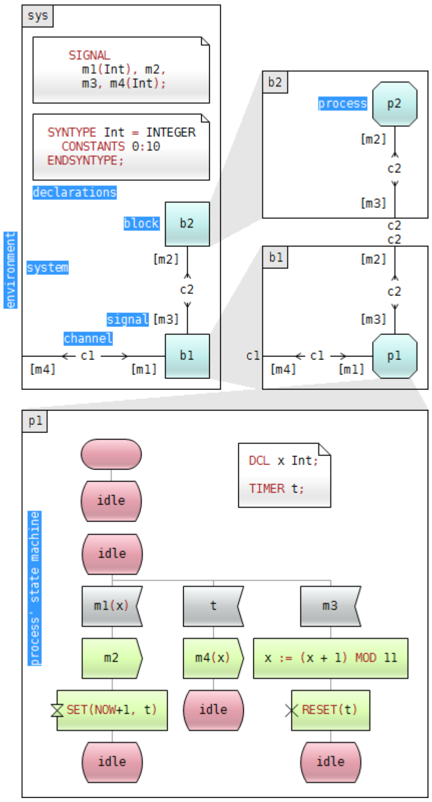


Fig. 2. Excerpt of a simple SDL model.

B. TTCN-3

The Testing and Test Control Notation Version 3 (TTCN-3) is a standardized testing technology developed and maintained by the European Telecommunication Standards Institute (ETSI). The ETSI TTCN-3 [3] recommendations are standardized by the ITU-T in the Z.160 series [4]. Figure 3 shows a TTCN-3 module definition with a single test case, where the system under test is the SDL example in figure 2.

```
module test {
  // Signals
  type record m1 { integer param1 }
  type record m4 { integer param1 }
  // Port
  type port c1_type message {
    out m1
    in m4
  }
  // Component
  type component sys {
    port c1_type c1
  }
  // Templates
  template m1 stimuli := { param1 := 5 }
  template m4 result  := { param1 := 6 }
  // Test-case
  testcase tc() runs on sys {
    c1.send(stimuli)
    alt {
      [] c1.receive(result) { setverdict(pass) }
      [] c1.receive { setverdict(fail) }
    }
  }
}
```

Fig. 3. Example of a TTCN-3 module definition with a single test case.

C. Simulation

While SDL allows to precisely define discrete event process oriented models with deterministic behavior, TTCN-3 complements these models with different scenario descriptions to generate a complete simulation model.

PragmaDev Studio is a set of tools that helps specifiers, developers, and testers to manage complexity in the development of today's systems. A key functionality of the tool-set is provided by the *PragmaDev Simulator* as shown in figure 4.

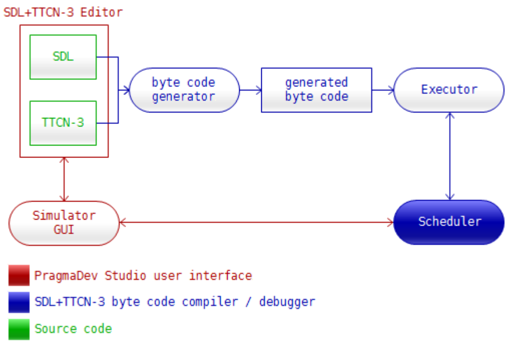


Fig. 4. Architecture of the PragmaDev Simulator.

The simulator allows execution of TTCN-3 test cases against an SDL system. SDL and TTCN-3 descriptions are translated into an internal representation (byte code) to be interpreted by the *executor*, which in turn forwards the scheduling of events to the *scheduler*.

The simulator is discrete event. It keeps a global queue for messages and the list of active timers. When the message queue becomes empty (i.e., all transitions triggered by the messages have been executed), the simulator jumps to the first timer in the list (if there is one). It then inserts a timer-expire message in the message queue, advances the current time to the expiration time of the timer, and removes the timer from the list.

IV. The JUICE Model

The JUICE simulation model is composed of two parts, i.e., the behavior and the configuration. The behavior is described in SDL, while the configurations in TTCN-3.

A. Behavioral Model

The behavior of the system is divided in three parts, and each of them is represented with an SDL process as shown in figure 5.

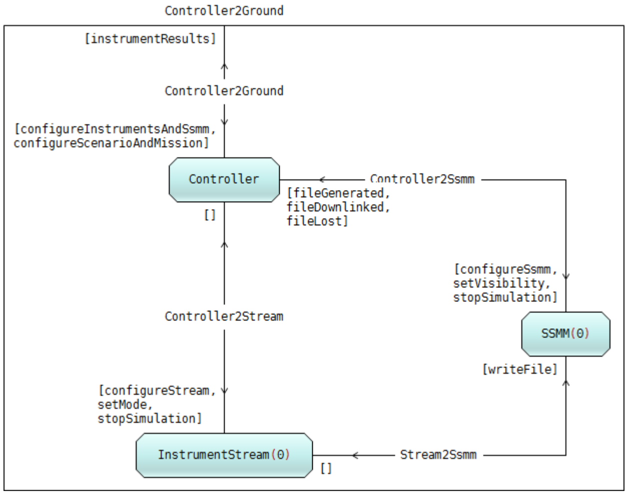


Fig. 5. SDL system for JUICE.

1) Controller: This process is responsible for initializing the instrument streams and the solid state mass memory (SSMM) based on the configuration received from the environment (i.e., the TTCN-3 test case). The configuration is received via the *configureInstrumentsAndSsmm* message as shown in figure 5. The *Controller* will create 40 instances of the *IntrumentStream* process (4 streams for each of the 10 instruments) and one instance of the *SSMM* process. It will also set the possible data rates for each instrument stream and the size of each directory of the SSMM. The *configureScenarioAndMission* message contains the changes in data rate for each instrument in time (called scenario), and the mission visibility periods when transmission to Earth is possible. The scenarios are presented in a (discrete) timetable as shown in table I.[1] The information is then forwarded to each *InstrumentStream* and the *SSMM* accordingly.

TABLE I
Example Scenario for Instruments

Time	GALA_1	JANUS_2	MAJIS1_3	MAJIS2_4	...
0	1	1	1	1	...
39505	1	1	2	1	...
54000	2	1	2	1	...

[1] For readability the data rate is represented by a single digit integer (called mode) specific to each instrument, e.g., for GALA_1 mode 1 means 0 kbit/sec and mode 2 means 6.5 kbits/sec.

The same logic is applied for the mission visibility, which is also presented in a timetable as shown in table II.

TABLE II
EXAMPLE OF MISSION VISIBILITY

Time	Visibility
0	false
54000	true
82800	false

SDL timers are used to ensure precise timing as shown in the tables. In a nutshell, a timer is started for the first row in the table where its timeout value is that shown in the table. When the timer expires the new data rates (modes) are sent to the *InstrumentStream* with the *setMode* message, and the change in visibility is sent to the *SSMM* with the *setVisibility* message. This is repeated for every row in the tables as shown in figure 6 with the timers *scenarioTimer* and *missionTimer*.

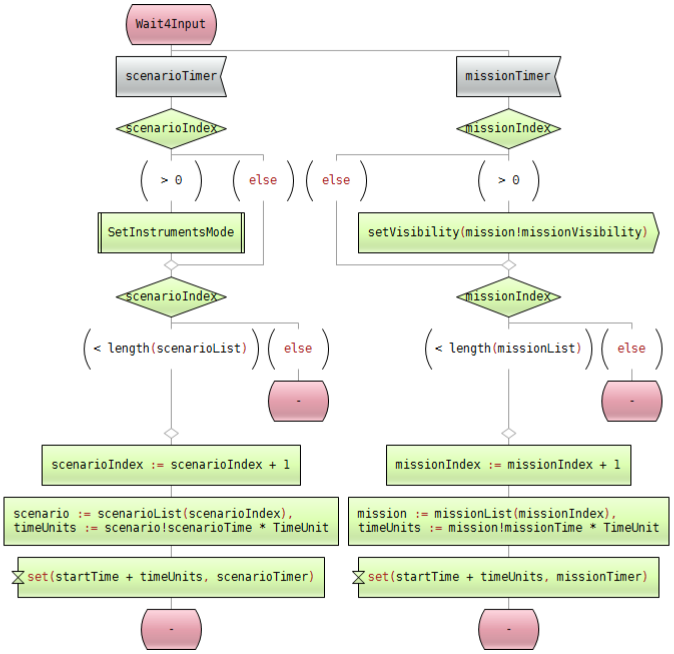

Wait4Input
scenarioTimer
missionTimer
scenarioIndex
missionIndex
> 0
else
else
> 0
SetInstrumentsMode
setVisibility(mission!missionVisibility)
scenarioIndex
missionIndex
< length(scenarioList)
else
< length(missionList)
else
-
-
scenarioIndex := scenarioIndex + 1
missionIndex := missionIndex + 1
scenario := scenarioList(scenarioIndex),
timeUnits := scenario!scenarioTime * TimeUnit
mission := missionList(missionIndex),
timeUnits := mission!missionTime * TimeUnit
set(startTime + timeUnits, scenarioTimer)
set(startTime + timeUnits, missionTimer)
-
-

Fig. 6. Excerpt from the behavior of the *Controller* process.

When done (simulation is finished) the *Controller* will sent the relevant results to the environment (TTCN-3 test case) with the *instrumentResults* message.

2) InstrumentStream: This process is responsible for generating streams of data and sending them to the *SSMM* for storage. The unit used for data is the file size in bytes. This is done for two reasons: (1) transmission to Earth is done when a file is ready (not just any amount of bytes), and (2) generating bits or bytes at the given data rates means generating a huge amount of events during simulation which degrades performance.

The behavior of the *InstrumentStream* is based on the file switching criteria given in section II. Two SDL timers are started, one for each criteria, i.e., *time* and *size*. While for the *time* criteria the value of the timer is straightforward, the *size* criteria involves some calculation. In this case the timeout value for the timer is computed using the *size* and the instruments current data rate. The first timer to fire will define the criteria to be used for setting the size of the file and sending it to the *SSMM* for storage with the *writeFile* message. This behavior is shown in figure 7.

The SDL *go* label in the figure points to the same behavior (not shown here), i.e., starting both timers (*sizeSwitch* and *timeSwitch*) and computing the size of the file to be sent. Size computation is straightforward using the *size* criteria. With the *time* criteria the size is computed using the time value and the current data rate of the instrument.

3) SSMM: This is the most complex process of the three in terms of behavior, because it is responsible for:

- storing files from instrument streams to their corresponding directory,
- transmitting files to Earth based on directory priority,
- applying error rates to the transmission,
- and in case of error(s), marking the concerned file(s) for top priority transmission in the next visibility period.

Storing a file in its corresponding directory is quite straightforward as shown in figure 8. When receiving a *writeFile* message from an *InstrumentStream*, the process will first check whether there is still free space in the directory corresponding to the instrument. If this is the case the file will be added to the list of files of that directory, and the *fileGenerated* message will be sent to the *Controller* which keeps track of the total number of generated files as a relevant output of the simulation. If the directory is full a *fileLost* message is sent to the *Controller*, which counts also the lost files due to not enough memory. This is another output of simulation relevant to architects whose objective is to reduce at minimum the amount of lost data.

Transmission is done in two bands, and each directory has an assigned band to it. When there is a change in visibility and transmission is possible, the *SSMM* will first check whether there are any files waiting for retransmission (due to previous errors). If this is the case then these files are sent in the order they appear in the retransmission queue, i.e., first-in-first-out. When the retransmission queue is empty then the transmission of the files in the directories can begin. For both bands the first file in a non-empty directory with the highest priority is selected for transmission. If there are more directories with the same priority, transmission is done in a round-robin fashion. The time needed for transmitting the file is computed using the file size and the bandwidth of the assigned band. The computed value is used in an SDL timer, and when it expires the file is considered to be transmitted. However, due to possible errors the file is not removed from the directory. If errors occur during transmission then the file is placed in the retransmission queue but is not removed from its directory. Only on complete and successful transmission, i.e., when the acknowledgement from Earth is received, the file is removed and the directory size updated.

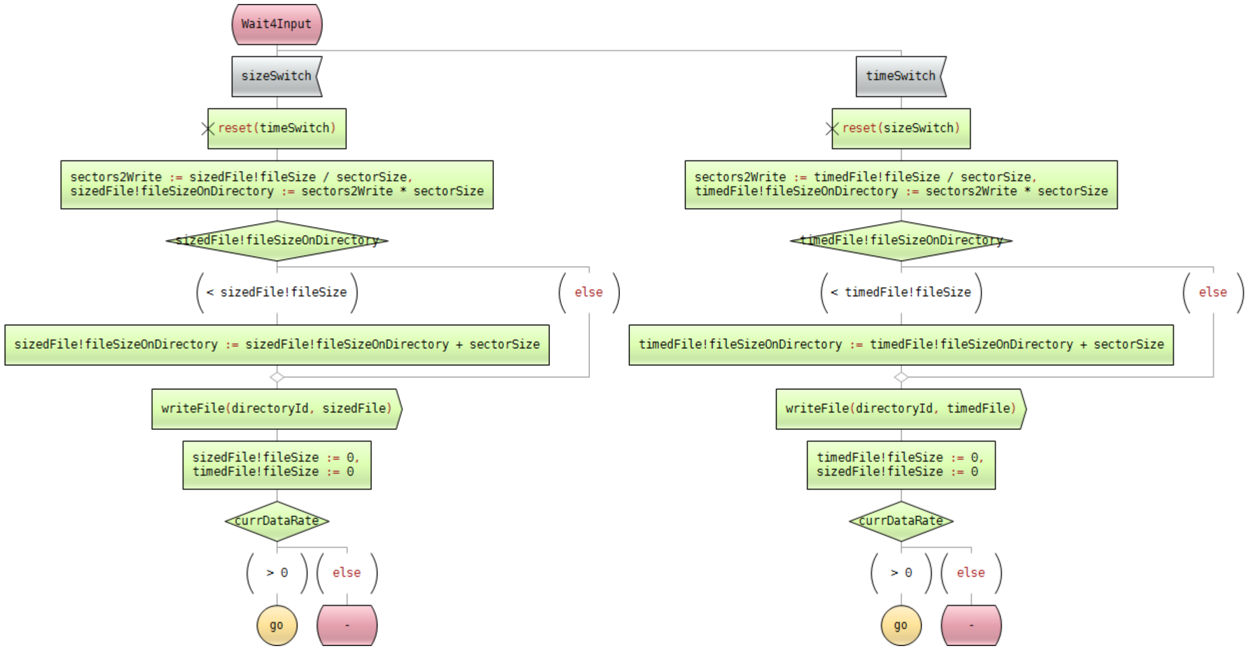

Wait4Input
sizeSwitch
reset(timeSwitch)
sectors2Write := sizedFile!fileSize / sectorSize,
sizedFile!fileSizeOnDirectory := sectors2Write * sectorSize
sizedFile!fileSizeOnDirectory
< sizedFile!fileSize
else
sizedFile!fileSizeOnDirectory := sizedFile!fileSizeOnDirectory + sectorSize
writeFile(directoryId, sizedFile)
sizedFile!fileSize := 0,
timedFile!fileSize := 0
currDataRate
> 0
else
go
-
timeSwitch
reset(sizeSwitch)
sectors2Write := timedFile!fileSize / sectorSize,
timedFile!fileSizeOnDirectory := sectors2Write * sectorSize
timedFile!fileSizeOnDirectory
< timedFile!fileSize
else
timedFile!fileSizeOnDirectory := timedFile!fileSizeOnDirectory + sectorSize
writeFile(directoryId, timedFile)
timedFile!fileSize := 0,
sizedFile!fileSize := 0
currDataRate
> 0
else
go
-

Fig. 7. Excerpt from the behavior of the *InstrumentStream* process.

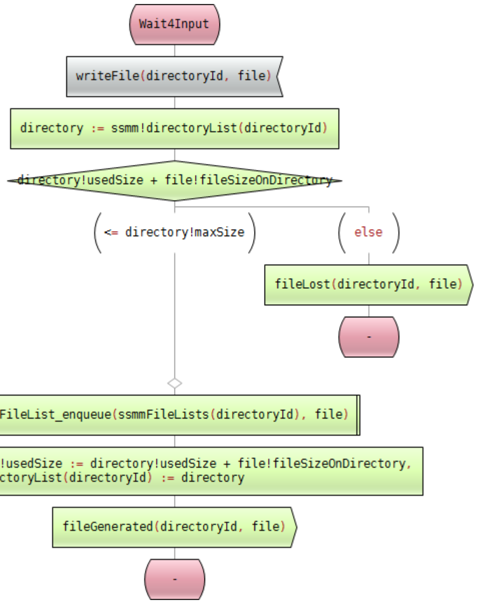

Wait4Input
writeFile(directoryId, file)
directory := ssmm!directoryList(directoryId)
directory!usedSize + file!fileSizeOnDirectory
<= directory!maxSize
else
fileLost(directoryId, file)
-
FileList_enqueue(ssmmFileLists(directoryId), file)
directory!usedSize := directory!usedSize + file!fileSizeOnDirectory,
ssmm!directoryList(directoryId) := directory
fileGenerated(directoryId, file)
-

Fig. 8. File storage in the *SSMM* process.

Figure 9 shows an excerpt from the *SSMM* behavior for handling retransmissions. The *PDU* (Protocol Data Unit) encapsulates information about the file to be retransmitted, and the retransmission queue is actually a list of PDUs. In a nutshell, the behavior shown in figure 9 consists in computing the time needed to transmit the file in the PDU, starting the *pduTimer* with the computed value, and removing the PDU from the list when the timer expires. The *resumePduTimer* is needed due to possible interruptions in the transmission from visibility changes.

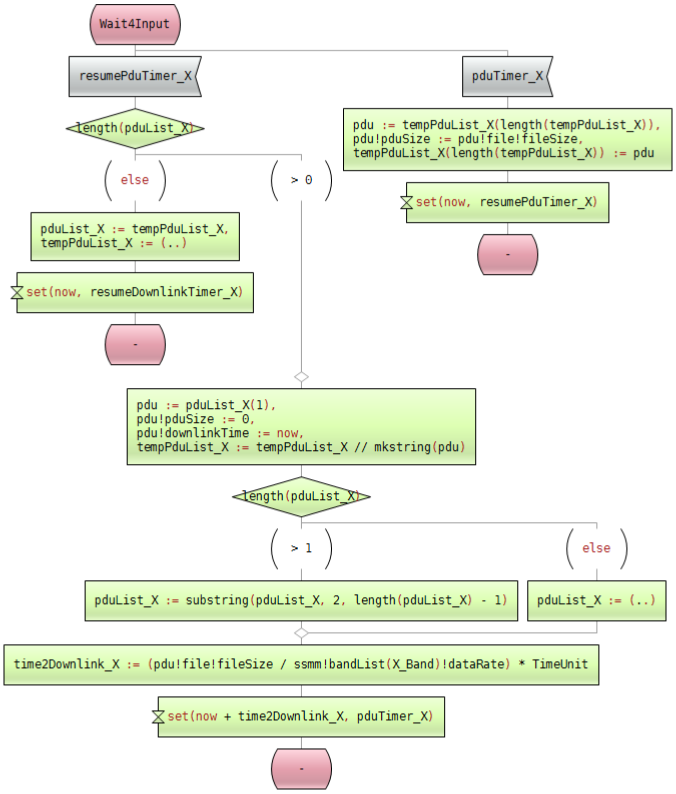

Wait4Input
resumePduTimer_X
length(pduList_X)
else
> 0
pduList_X := tempPduList_X,
tempPduList_X := (..)
set(now, resumeDownlinkTimer_X)
-
pduTimer_X
pdu := tempPduList_X(length(tempPduList_X)),
pdu!pduSize := pdu!file!fileSize,
tempPduList_X(length(tempPduList_X)) := pdu
set(now, resumePduTimer_X)
-
pdu := pduList_X(1),
pdu!pduSize := 0,
pdu!downlinkTime := now,
tempPduList_X := tempPduList_X // mkstring(pdu)
length(pduList_X)
> 1
else
pduList_X := substring(pduList_X, 2, length(pduList_X) - 1)
pduList_X := (..)
time2Downlink_X := (pdu!file!fileSize / ssmm!bandList(X_Band)!dataRate) * TimeUnit
set(now + time2Downlink_X, pduTimer_X)
-

Fig. 9. Excerpt from the file transmission part of the *SSMM* process.

B. Configurations

Configurations are written in TTCN-3 and sent to the SDL model. A configuration has two parts: initial and runtime. The initial configuration consists of sending the available modes (data rates) to each instrument (4 streams per instrument), and configuring the SSMM as shown in figure 10.

```
const configureInstrumentsAndSsmm config1 := {
  modeList := {
    // Instrument 1
    {
      {0, 6.5, 0, 0}, // Stream 1
      {0, 6.5, 0, 0}, // Stream 2
      {0, 6.5, 0, 0}, // Stream 3
      {0, 6.5, 0, 0}  // Stream 4
    },
    // Instrument 2
    {
      {0, 750, 300, 547.5},
      {0, 750, 300, 547.5},
      {0, 750, 300, 547.5},
      {0, 750, 300, 547.5}
    }
    // ...
  },
  ssmm := {
    fileListCapacity := 100000,
    segmentSize := 4.096,
    sectorSize := 16384,
    bandList := {
      {21.8, 100}, // X band
      {13, 10}     // Ka band
    },
    directoryList := {
      // Instrument 1
      {1000000000, 0, 3600, X_Band, 1, 0},
      {1000000000, 0, 3600, X_Band, 2, 0},
      {1000000000, 0, 3600, X_Band, 3, 0},
      {1000000000, 0, 3600, X_Band, 4, 0},
      // Instrument 2
      {1000000000, 65536, 0, Ka_Band, 1, 0},
      {1000000000, 65536, 0, Ka_Band, 2, 0},
      {1000000000, 65536, 0, Ka_Band, 3, 0},
      {1000000000, 65536, 0, Ka_Band, 4, 0}
      // ...
    }
  }
}
```

Fig. 10. Example of a TTCN-3 module definition with a single test case.

A runtime configuration consists of a scenario for instruments and mission visibility as shown in figure 11. These are the TTCN-3 representations of the information given in table I and II. A scenario contains the modes of all instruments in time. A mode is an integer from 1 to 4, i.e., the index for the instrument streams in the *modeList* in figure 10, where the values are the possible data rates for the stream.

```
const ScenarioListType scenario1 := {
  {     0, {1,1,1,1,1,1,1,1,1,1,1,1,1}},
  { 39505, {1,1,1,1,2,1,2,1,1,2,1,1,1}},
  { 54000, {1,1,1,1,2,1,2,1,1,2,1,1,1}}
  // ...
}

const MissionListType mission1 := {
  {     0, false},
  { 54000, true},
  { 82800, false}
  // ...
}
```

Fig. 11. Example of a TTCN-3 module definition with a single test case.

The initial and runtime configuration are sent to the SDL system with the *configureInstrumentsAndSsmm* and *configureScenarioAndMission* messages (shown in figure 5). The SDL system will reply back to the TTCN-3 at the end of simulation with the *instrumentResults* message.

V. Results

The JUICE SDL model and the TTCN-3 configurations are run in co-simulation mode in the PragmaDev Simulator. The results of a co-simulation are test case verdicts, e.g., pass, fail, etc. This can be useful in defining certain conditions on the relevant outputs and ensuring they are met, e.g., if there are more than 100 files lost then set the verdict to fail. However, the desired result of a simulation is the outputs themselves rather than "guards" on them. The only mechanism in TTCN-3 to output data is the *log* function, which prints text to the standard output. Although there are ways to further manipulate this information outside TTCN-3, the objective is to present to architects all relevant information in the most efficient way. For this we defined and implemented *PragmaLib*, a TTCN-3 module integrated into the simulator. The module introduces file manipulation and display of tables and graphs in TTCN-3 via functions, where graphical display of tabular data follows the concepts introduced in [5]. Relevant outputs are presented in a table and radar graph as shown in figure 12.

The advantage of this kind of presentation is that multiple configurations (TTCN-3 test cases) can be run, and their results displayed in the same graph. This facilitates comparison and decision on the optimal configuration. The rows in the tables are the instruments, while the columns are the different configurations. The values in the table are the relevant outputs requested by the architects:

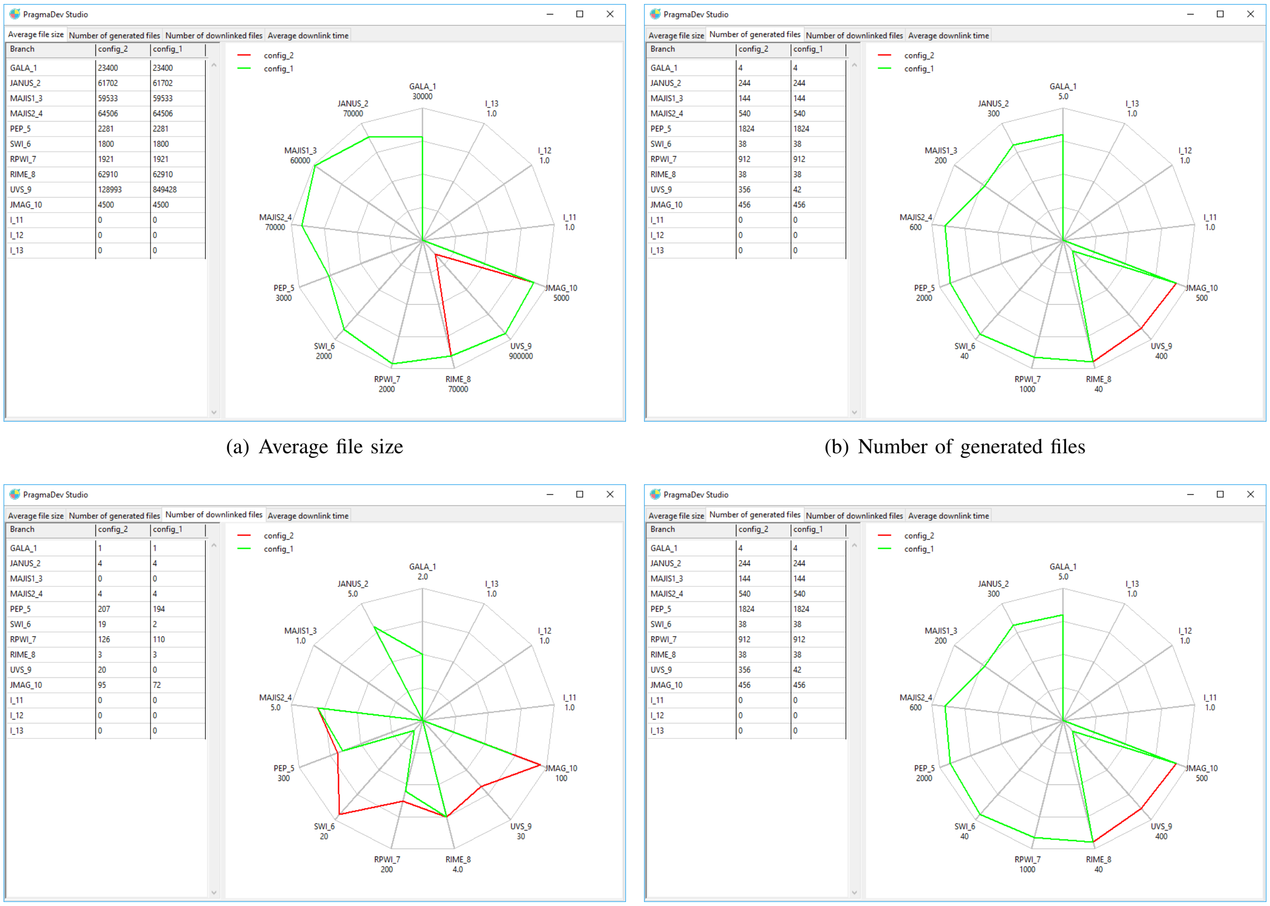
- average file size generated by the instrument streams in figure 12(a),
- total number of generated files by the instrument streams in figure 12(b),
- number of successfully transmitted files for the instrument in figure 12(c), and
- average time of all transmitted files for the instrument in figure 12(d).

The second group of functions in the *PragmaLib* module allows users to read and write external files, features which are not available in standard TTCN-3. The reason for introducing such functions is that architects usually manipulate and store configuration parameters and simulation results in spreadsheet applications. The file manipulation functions facilitate the import from these application into TTCN-3 to reduce manual coding. Also, the results displayed in figure 12 can be easily exported to a spreadsheet application for further manipulation, presentation, or storage.

VI. Related Work

Initial requirements and design engineering of new products are key for mastering cost, schedule, quality, and risks of the avionics development [6]. In this context, model-based techniques are providing viable solutions to the problems the aerospace domain is faced with [7].

A model based development approach and tooling focusing on data is presented in [8]. The approach is based on SysML and SCADE system (a domain-agnostic system modeling language), and has been applied to an avionics system case



Branch	config_2	config_1
GALA_1	23400	23400
JANUS_2	61702	61702
MAJIS1_3	59533	59533
MAJIS2_4	64506	64506
PEP_5	2281	2281
SWI_6	1800	1800
RPWI_7	1921	1921
RIME_8	62910	62910
UVS_9	128993	849428
JMAG_10	4500	4500
I_11	0	0
I_12	0	0
I_13	0	0

(a) Average file size

Branch	config_2	config_1
GALA_1	4	4
JANUS_2	244	244
MAJIS1_3	144	144
MAJIS2_4	540	540
PEP_5	1824	1824
SWI_6	38	38
RPWI_7	912	912
RIME_8	38	38
UVS_9	356	42
JMAG_10	456	456
I_11	0	0
I_12	0	0
I_13	0	0

(b) Number of generated files

Branch	config_2	config_1
GALA_1	1	1
JANUS_2	4	4
MAJIS1_3	0	0
MAJIS2_4	4	4
PEP_5	207	194
SWI_6	19	2
RPWI_7	126	110
RIME_8	3	3
UVS_9	20	0
JMAG_10	95	72
I_11	0	0
I_12	0	0
I_13	0	0

(c) Number of downlinked files

Branch	config_2	config_1
GALA_1	4	4
JANUS_2	244	244
MAJIS1_3	144	144
MAJIS2_4	540	540
PEP_5	1824	1824
SWI_6	38	38
RPWI_7	912	912
RIME_8	38	38
UVS_9	356	42
JMAG_10	456	456
I_11	0	0
I_12	0	0
I_13	0	0

(d) Average downlink time

Fig. 12. Graphical presentation of simulation results with the PragmaDev Simulator.

study. The authors argue for the benefits of the model based approach by demonstrating a complete flow from functional architecture capture down to platform deployment.

In the the ESA e.Deorbit study, a model-based process for system modelling and simulation has been developed for supporting the iterative generation and maturation of the system requirements, architectures and system budgets [9]. The approach links SysML, CDP, Matlab, and ModelCenter in one process for implementing the design-analysis-verification workflow [6].

The approach presented in this paper demonstrates that SDL and TTCN-3 do provide efficient means for modelling and simulation of aerospace systems for optimization purposes. Efficiency comes from the event based nature of the system, modeling languages, and simulation engine, while the solutions mentioned above use cycle based simulation. Furthermore, TTCN-3 being a testing language, enables automatic checking of simulation results, which is an added value that facilitates decision making.

VII. Conclusion

The JUICE use case presented in this paper is a typical process based event driven system, and as such the use of SDL and TTCN-3 is a sound choice. However, modeling is not free of challenges.

Although an event driven simulation model per se performs very well in terms of execution time, there are some tradeoffs and pitfalls to look out for. Care should be taken in the number of events generated, because there is a risk for the model to become more time driven rather than event driven. We were faced with this issue when modeling the instrument streams. Modeling the data rate of the instrument in the given unit (i.e., kbits/sec) generated a huge amount of events to the point where execution times were not acceptable. However, as instrument readings were stored in files (and transmission was also based on files), it made sense to build the model around the file as a unit for defining events. This reduced execution time significantly without losing precision in the results, but required additional changes to the model. As an example, the case of a mode (data rate) change while writing into the file had to be handled explicitly.

Another challenge we were faced with was the interfacing with the potential users of our solution. We noticed that users experienced with spreadsheet applications for decision making would have to acquire coding skills for inputting configurations and especially for retrieving and presenting simulation results. To aid them in this regard we introduced file manipulation and graphical display of data to our existing technologies.

The JUICE use-case using the PragmaDev tool suite comforts Airbus Defence & Space on the capability to use model based formal techniques to validate new and complex operational strategies. The objective is not to model all the system for each program but rather to address specific problems early in the design phase. (1) Rapid-prototyping capability, (2) good performance and scalability of tools as well as, (3) mature results visualization for analysis are three main drivers to ensure industrial appropriation of such techniques. Airbus Defence & Space will continue exploration of the model based approach.

References

[1] European Space Agency, "ESA Science & Technology: JUICE," http://sci.esa.int/juice, 2016.

[2] ITU-T, "Specification and Description Language – Overview of SDL-2010," International Telecommunication Union – Telecommunication Standardization Sector, ITU-T Recommendation Z.100, 2016, http://handle.itu.int/11.1002/1000/12846.

[3] ETSI, "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language," European Telecommunications Standards Institute, ETSI Standard ES 201 873-1, 2014, http://www.ttcn-3.org/index.php/downloads/standards.

[4] ITU-T, "Testing and Test Control Notation version 3: TTCN-3 core language," International Telecommunication Union – Telecommunication Standardization Sector, ITU-T Recommendation Z.160, 2014, http://handle.itu.int/11.1002/1000/12346.

[5] M. Brumbulli and E. Gaudin, *Optimizing Performance of SDL Systems*. Springer International Publishing, 2016, pp. 100–115.

[6] A. Rossignol, S. Estables, A. Cortier, and D. Thomas, "Model based Avionics roadmap and case studies," https://indico.esa.int/indico/event/148/session/5/contribution/41/material/slides/0.pdf, 2016.

[7] European Space Agency, "Applying MBSE to a space mission," http://blogs.esa.int/cleanspace/2017/08/28/applying-mbse-to-a-space-mission, 2017.

[8] T. Le Sergent, F.-X. Dormoy, and A. Le Guennec, "Benefits of Model Based System Engineering for Avionics Systems," in *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, 2016.

[9] S. Estable, T. Granger, T. Zoebelein, N. Brauer, T. Lochow, I. Tolchinsky, and S. Genere, "Systems Modelling and Simulation of the ESA e.Deorbit Space Debris Removal Mission," https://www.phoenix-int.com/tech_papers/systems-modelling-simulation-esa-e-deorbit-space-debris-removal-mission, 2017.