Modelica, FMI and SSP for LOTAR of Analytical mBSE models: First Implementation and Feedback

Clément Coïc[1] Adrian Murton[2] Juan Carlos Mendo[3] Mark Williams[3] Hubertus Tummescheit[1] Kurt Woodham[4]

[1]Modelon, Sweden, {clement.coic, hubertus.tummescheit}@modelon.com

[2]Airbus Operations Ltd. United Kingdom, adrian.murton@airbus.com

[3]The Boeing Company, USA, {juan.c.mendo, mark.williams}@boeing.com

[4]NASA Langley Research Center, USA, kurt.woodham@nasa.gov

Abstract

LOng Time Archiving and Retrieval (LOTAR) of models is key to using the full capabilities of model-Based System Engineering (mBSE) in a system lifecycle – including certification. The LOTAR MBSE workgroup is writing the EN/NAS 9300-Part 520 to standardize the associated process, in the aeronautics industry, and suggests the usage of Modelica, FMI and SSP standards for its purpose. Acceptance of such a process requires a match between industrial needs and software vendor implementations. This is helped by a tool-agnostic implementation of the process and following specific adaptations within the Modelon Impact software. This initiative – inside the LOTAR workgroups – highlights the suitability of such a process but also points at flaws or overhead due to the lack of connection between the Modelica, FMI and SSP standards, as well as the MoSSEC (ISO 10303-243) standard. The recommendations proposed in this document could have a significant impact on the final adoption of the LOTAR standard – relying on Modelica, FMI and SSP standards.

Keywords: Archiving, Retrieval, LOTAR, mBSE, MoSSEC, FMI, SSP

1 Introduction

Contrary to the software industry where end-of-life is programmed and conversion to a newer alternative is "enforced," the industrial products containing complex cyber-physical systems have longer lifecycles and associated maintenance.

In the aerospace industry, designing an aircraft takes about a decade. The production cycle averages three decades, and their service life extends for another three decades. Deciding to develop a new aircraft is a choice that impacts two thirds of the next century. The technology and design choices, the system and component sizing, the rationale and arguments in each decision taken shall be stored and kept accessible during the aircraft's entire lifecycle. This enables its potential evolutions and design reuse opportunities. It capitalizes on the work and knowledge and supports a response to future questions. Other key aspects of data archiving and retrieval in the aerospace industry is to provide a basis for the certification of future modifications, address component obsolescence, and support accident investigations.

These are the challenges that the LOTAR international consortium of Aerospace manufacturers –jointly facilitated by AIA, ASD-Stan, AFNeT, prostep ivip and PDES, Inc. – are facing through the creation and deployment of the EN/NAS 9300 series of standards for long-term archiving and retrieval of digital data. To ensure industry adoption, the resulting process must be based on standardized practices and proven solutions, listed on the website (LOTAR International 2021). The authors of this papers are active members of the "MBSE workgroup" within the LOTAR consortium.

Model-Based System Engineering (MBSE) definition, *"the formalized application of modeling to support system requirements, design, analysis, verification and validation beginning in the conceptual design phase and continuing throughout the development lifecycle"* (INCOSE Operations 2007) is widely used in the aerospace industry.

While MBSE includes all types of models, the authors previously introduced the acronym "mBSE" [or "little m BSE" as opposed to the "big M BSE"] to narrow the focus on the preservation of system-descriptive and analytical models that are explicit, coherent, and consistent (Nallon 2021). The integrated models provide high-fidelity, rich representations – potentially of different granularity of sub-systems. These models are viewpoints that support the decisions affecting the product's architecture, new technologies, or component sizing. This distinction is necessary to separate this work from other types of models, e.g. 3D CAD models. (This does not mean that these models should be decoupled.)

The archiving and retrieval of the models developed is mandatory to utilize the efforts and rationales the model served throughout the lifecycle of the (cyber-physical) system it represents. mBSE data is also applicable to the certification process and in-service maintenance of Aerospace products. The need for long-term archiving is an existing regulatory requirement. As many design representations shift to a digital format the urgency of defining archiving standards is in response to a critical industry need. EN/NAS 9300 Part 520 standardizes the long-term archiving and retrieval process for analytical mBSE models.

Section 2 of this paper concisely introduces the suggested steps for archiving and retrieval and discusses the supplemental needs of the data archive and model manifest. Section 3 presents a tool-agnostic implementation and its integration within Modelon Impact. Section 4 discusses the prototype results and proposes specific recommendations for the standards being used.

2 Archiving and retrieval process

2.1 Abstract and Keywords

While the EN/NAS 9300 Part 520 describes the archiving and retrieval process in more detail, the main points are listed below.

Archiving

- Develop and validate an mBSE model,
- Create an associated meta-data manifest,
- Export the model as an FMU or SSP,
- Include the manifest in the “extra” folder of the FMU or SSP
- Archive the FMU or SSP, together with its manifest, in the archiving platform/repository,
- Populate the AIP (Archive Information Package) with information from the manifest

Retrieval

- Access the AIPs on the archiving platform
- Select the desired archived FMU/SSP by examining the repository’s AIPs
- Retrieve the FMU/SSP
- Consult the associated manifest to validate the retrieval results
- Verify that the model is not corrupt

For this standard to be easily deployable, the emphasis is on the archiving and retrieving process of the model, not its creation. A tool vendor is welcomed and even encouraged to implement some of these steps earlier in the modeling process. In a typical scenario, the model manifest is populated early in the model’s lifecycle. The population process is typically iterative throughout the product design phase and could be optimized to support additional goals such as model exchange. These steps will typically happen prior to the model’s export as an FMU for archival purposes. However, as long as the consistency of the model and the meta-data is sustained, the order of operations is not imposed.

2.2 Relying on existing standards

The LOTAR MBSE workgroup made an extensive effort to reference and map most related standards, their applicability, usage, and maintenance (Williams 2021). One aim was to define which standards to rely on for the archive and manifest formats. The first consideration was a neutral format with the widest potential tool support for long term archiving. The second consideration was endorsement by the Aerospace OEMs. It was found that:

- The FMI standard has reached a level of maturity and availability that supports model archiving and preservation.
- The SSP standard – being the structured system variant of the FMI standard – is also a recommended alternative for system model archiving and preservation.
- The ISO STEP AP243 (MoSSEC 2021) standard, in a format similar to the Model Identity Card (MIC), is the recommended format for the model manifest.

These mature tool-agnostic standards form a solid base for the Part 520 standard on which any tool vendor can build a marketable solution.

2.3 LOTAR manifest

The LOTAR manifest is the identity card that enables each model to travel and be identified uniquely. In the first tool-agnostic implementation presented by this paper the manifest is stored as an XML (W3C 2006) file with tags arranged in various categories, which gather attributes and associated values.

As represented in Listing 1., the simplicity of the XML mark-up language makes it a suitable candidate for parsing of the manifest or performing further action on it – e.g. generation of a graphical representation, or populating repository attributes.

Listing 1. A short extract from a sample manifest:

```
<LOTAR_Manifest>
  <GeneralPLM>
...
  </GeneralPLM>
  <DevelopmentIntegrationAndExecution>
...
  </DevelopmentIntegrationAndExecution>
  <PhysicsContentAndUsage>
    <PhysicsContentProperties
      Dimension=""
      PhysicsDomain=""
```


DOI 10.3384/ecp2118149

```
        Timescale=""
        Linearity=""
        ModelType_Usage="">
      </PhysicsContentAndUsage>
      <ValidityRange
        ValidityRange="">
      </ValidityRange>
      <ModelFidelity
        RepresentedPhenomena=""
        NeglectedPhenomena="">
      </ModelFidelity>
    </PhysicsContentAndUsage>
...
    <ModelVariables
    </ModelVariables>
    <VerificationAndValidation
...
    </VerificationAndValidation>
</LOTAR_Manifest>
```

The different categories in the manifest aim to capture the design intent of the model (what), the rationale and purpose for creating the model (why), the content, fidelity, and format of the model (what/how), as well as its provenance (who/when).

These specific metadata categories can be mapped to ISO STEP AP243 (MoSSEC), and they are used to capture the systems engineering context around each model to be archived. The MoSSEC specification standardizes such context. Sharing or archiving, the MoSSEC-styled metadata information facilitates every model's availability, retrieval, and reusability

3 First implementation

The LOTAR mBSE workgroup solicited Modelon to implement a tool-agnostic version of the archive and retrieval process. The aim was to detect the "paper cuts" in the process that could hinder the standard's adoption. Performing this type of early prototyping, prior to the standard's initial release, verifies the viability of the identified mBSE data standards and the associated workflow needed for archiving, sharing, and retrieving analytical models.

3.1 Definition of the prototype system

As this work focuses on the archiving and retrieval process, the prototype system is made as simple as possible while still illustrating the optional process steps. The LOTAR mBSE workgroup selected the Regulated Actuator system represented in **Figure 1** for their proof of concept.

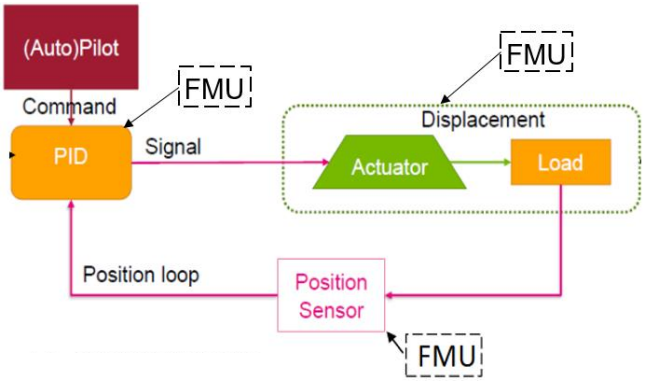

Figure 1. Regulated Actuator System

The diagram represents a system whose goal is to control the position of a flight control surface to simplify the (auto) pilot's response when facing external loads. The system is composed of three subsystems:

- The controller, a simple PID (proportional, integral, derivative) controller that receives the (auto) pilot position commands and the measured actuator position. It then outputs a command based on the errors detected between both.
- The plant model gathers both the actuator and load models. An output is the actual surface position.
- A sensor model inserts a delay in the measurement.

The three models are highly simplified representations of the physics involved. The focus is to define a common process independent of the tools and potentially different environments that could be used by different teams. Furthermore, each subsystem is modeled separately and exported as an FMU; the overall system can be exported as a SSP – by composition of the previously built FMUs.

3.2 A tool-agnostic implementation

3.2.1 Modelica models and FMU generation

The three models described in section 3.1 and presented in **Figure 1** are developed using the Modelica language. Once more, the reason is not to take advantage of the language capabilities but to take advantage of its openness. Indeed, the models are deliverables to the prototyping effort and can be shared in text format for further use. A derived advantage is the inclusion of the documentation-annotation provisions in MLSv35r0 (Modelica Association 2021) that enables embedding the model description and experimental frame, within the model itself, in HTML format.

The FMUs (Functional Mockups Units) are generated using the Modelon compiler included in Modelon Impact but could have been generated by any other Modelica compiler, e.g., OpenModelica. They have been generated in both the Model Exchange and Co-Simulation formats for further study and use.

Note: for LOTAR purposes, a co-simulation FMU has a clear advantage because the solver is stored within the FMU – by definition. This is a highly relevant point for long term archiving. Which solvers will be available in 50 years from now? How will one be able to couple them with another FMU? Nevertheless, the model exchange FMU also has obvious advantages because they are great candidates for a direct coupling into an SSP prior to archiving. The results of this prototype will be analyzed by the LOTAR team to understand the preferred archive alternatives, and the best potential formats for archiving. A Modelica language tool may not always provide the source, and entire repositories of archived formats may need to be converted to alternative standards in the future.

The following steps of the process are achieved by relying on Python scripts – using open-source or, at least, free of use packages. As a reminder, this is an implementation that helps to bring the standard to life but not the standard itself – anyone is free to implement it using different means.

3.2.2 Creation of the LOTAR manifest

The LOTAR manifest being written is an XML file (W3C 2021), created using the Python module xml.etree.ElementTree. The XML tree structure is built first and then attributes are set to their values. Unknown values are left as empty strings.

In the future envisioned process, populating the manifest metadata should be a semi-automatic process using a tool specific implementation that occurs prior to the start of the archival process.

However, the EN/NAS 9300 Part 520 standard has not been formally released yet, so this is currently performed manually. Nevertheless, this first implementation highlighted many commonalities with the "modelDescription,xml" file contained in the FMUs. The PyFMI (PyFMI 2020) package is used to load and interact with the FMU in its most basic form: accessing the modelDescription information. This way, many fields of the manifest are automatically populated by the Python script. A simple extract is defined in Listing 2.

Listing 2. A short extract from a sample manifest:

```
# Root
manifest=ET.Element('LOTAR_manifest')
# LOTAR_manifest > GeneralPLM
GeneralPLM=ET.SubElement(manifest,'GeneralPLM')
# LOTAR_manifest > GeneralPLM > ...
ProvenanceOwnershipDate=ET.SubElement(GeneralPLM,
'ProvenanceOwnershipDate')
# Populate creation date
ProvenanceOwnershipDate.set('Created_on',
model.get_generation_date_and_time())
```

Note that another option would be to extract and parse the XML file directly. This would be, however, more laborious and the solution presented relies on maintained python packages, and thus is more convenient.

At this point, it becomes important to note that a subset of model manifest fields can be taken straight from the "modelDescription.xml". However, the model manifest offers extra metadata that travels with the original model, regardless of its format: FMU or native.

3.2.3 Inclusion of the manifest in a Functional Mockup Unit

Because the FMU is a zip file, the inclusion of the manifest can be performed automatically with a Python module such as zipfile (Pyhton Software Foundation 2021). Three specific points are listed here:

- To prevent conflicts, the naming of the manifest uses reverse domain notation such as `extra/org.mossec/LOTAR_Manifest.xml`
- Ensure the manifest is added in the "extra" folder of the FMU recently available FMIv2.0.2 (Modelica Association, 2020) in the standard.
- To minimize the file corruption risks, it is recommended to open the FMU in a mode in which it is only possible to append new files, not to modify existing ones.

The FMU is now ready for archiving on the platform.

3.2.4 Accessing the manifest

Once the FMU is archived, the manifest should be accessible by the repository, so the contained information enables the user to identify the correct model needed for retrieval. The selected implementation consists of inverting the previous process: open the zip in read only mode, extract a copy of the manifest in a preselected location in the repository and then access this copy. This can be achieved with the same Python module zipfile.ZipFile.

Note that one recommendation for the repository could be to perform this manifest manipulation when archiving the FMU, keeping a pointer toward the original source file – thus collecting an organized set of manifests. Retrieval could then consist of browsing the set of stored manifests and selecting the correct one. The archiving platform could then access the related FMU, verify that the same manifest is included and perform the retrieval. From a LOTAR perspective, the manifest is very similar to the AIP mentioned previously and could be replicated accordingly. This would ensure the model metadata would exist both internally and externally to the FMU zipfile.

3.2.5 Repeating the process at system level

The prototype was designed so that a system level example could be studied and refined. The regulated actuator model is built as an SSP by composition of the existing FMUs. The manifest template, as defined in the first version of the EN/NAS 9300-Part 520, applies primarily to subsystem level component models and is

DOI 10.3384/ecp2118149

not directly applicable yet to system level simulations or setups. The Python scripts were adapted to reach the manifests inside the FMU zip files when archiving. In the future, the manifest of a structured system model should be defined and stored in the "extra" folder of the SSP – for consistency.

3.2.6 Availability of developed models and code

The LOTAR MBSE workgroup and Modelon agreed to make these models and Python code available to the LOTAR and eventually to the broader community. This is an added benefit of this first implementation: publicly available basic models and Python scripts – relying on open-source or free of use packages – that follow and implement the process from the draft Part 520 standard. This way, there are no proprietary restrictions preventing a tool vendor from implementing the standard. Modelon illustrates this fact with the addition of a custom function within Modelon Impact to write and add the manifest when exporting an FMU.

3.3 A tool-specific implementation

3.3.1 Modelon Impact in four sentences

Modelon Impact is a cloud native based modeling and simulation environment relying on and enhancing open technologies such as Modelica, FMI or Python (Modelon 2020). Modelon Impact is aimed at democratizing simulation to a broader audience by providing a user friendly, yet powerful interface to develop and simulate models or utilize them in a very narrow context, including the use of web applications (Coïc 2020a). Modelon Impact offers both steady-state and dynamic simulation capabilities (Coïc 2020b), which exposes more opportunities to the model developer to make a model best suited to the model user. Finally, it is possible to implement user specific workflows in Modelon Impact through Python-based custom functions which can interact with the Modelon Impact API.

3.3.2 A custom function for the manifest generation

One convenient way to implement a Modelon Impact specific implementation of this process is to develop a dedicated custom function that would write and add the manifest to the FMU when compiling a model.

The tool-agnostic prototype is Python-based and relying uniquely on open-source technologies. The step to implement a Modelon Impact custom function is thus minor as these also rely on the Python language.

Custom functions are available in Modelon Impact by hovering over the simulate button (see **Figure 1**) or by selecting the dedicated function in the experiment mode (see **Figure 2**). In the latter case, more actions are possible through user inputs. For example, it was decided to add the "Author name" and "Organization" as a user input fields so that this information would always be added to the manifest.

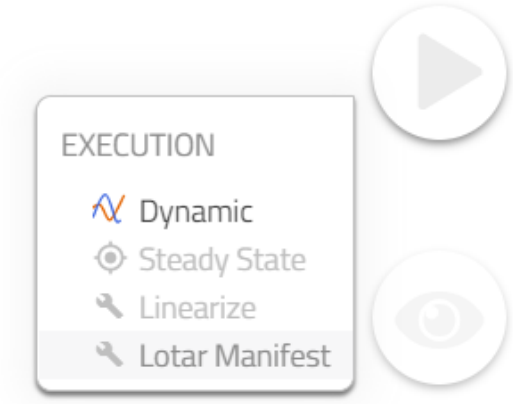

Figure 1. Direct generation of LOTAR manifest in Modelon Impact

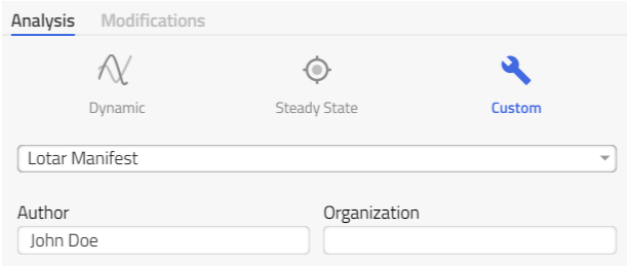

Figure 2. Generation of the LOTAR manifest after optional user input provision, in Modelon Impact

Notes:

- This is hard to demonstrate in a paper, but a short demonstration of the python interface using PyFMI (PyFMI 2020) is available.
- The rest of the process is independent of the modeling and simulation environment but occurs on the archiving platform. Therefore, Modelon limits their tool-specific implementation to this step.

4 Discussions and recommendations

The completion of this prototype, prior to the release of the Part 520 process standard, brought great insights on the applicability of the representative workflow and the potential need for future improvements. The following recommendations and criticisms concern both the Part 520 and the Modelica, FMI and SSP data standards. This paper is directed toward the users of these technologies.

4.1 Self-criticism

Constructive criticism is what ensures quality of work. This prototype was developed with this in mind: stress-test the process in order to spot any inconvenient or non-finalized steps. Listed below is a list of items that were identified as sources of improvement. The items are tagged with "minor" and "major" keywords to highlight their criticality. However, a "minor" item is not irrelevant as in the long run it could be a paper cut that prevents the standards future adoption.

- [Major] Create a proper mapping between the FMI "modelDescription.xml" and the LOTAR manifest. When filling out the manifest, many items were extracted from the PyFMI API. While this is convenient, this also shows some redundancies. A proper mapping would highlight whether the manifest should become a small extension of the existing modelDescription or remain a separate file.
- [Major] Define the content of the manifest for an SSP. Many of the fields identified for the FMU LOTAR manifest are less relevant at system level – especially if each FMU contains its own manifest. For example, what is the value of identifying the physical domains involved in an SSP if each FMU specifies its own? A substantial effort is expended by the industrials, under the guidance of prostep ivip to transform the "Glue Particle" into a data standard. This should expand the investigation of adding descriptive metadata to an SSP.
- [Major] How to precisely specify the format of the validation and verification scenarios. The LOTAR manifest includes these attribute fields but does not constrain their format. Specifying the file type for reference results would be a key to future success – especially when the validation tests are performed several decades later after retrieval from an archive.
- [Minor] Update the "Required" fields of the manifest. The manifest identified some fields as required (i.e. mandatory information) but the current version is not easily adapted to the FMI standard. For example, a field for "TargetTool_Name" is required while one benefit of the FMI standard is to be tool independent. Nevertheless, this field is relevant if some dedicated tests were performed in the future targeting a specific tool. This information could be maintained as a reference (although not "required").
- [Minor] Harmonize hierarchy and naming convention. Both "snake_case" and "camelCase" are used in the current LOTAR manifest sample. Some attributes repeat words in their names. This redundancy could be avoided by employing an additional hierarchical layer. This change is necessary to make the manifest more "attractive" for users and to remove sources of errors by using a formal naming convention.
- [Minor] Investigate how to add additional metadata to the manifest. It is expected that companies will need to define their own specific metadata that they will need to store within the manifest. This could be achieved in several different ways – e.g. by adding new attributes to existing tags or by defining new dedicated tags (for example, an "extra" tag?). This recommendation would be easy to implement.

4.2 Recommendations on used standards

The LOTAR MBSE workgroup remains confident that FMI and SSP standards provide a solid basis for the Part 520 standard. This makes the following constructive criticism even more relevant, as any improvement on these standards and/or associated tool implementations would also benefit the Part 520 indirectly. The criticality tags are also used here.

- [Major] Provide user entries for relevant metadata fields. When compiling an FMU or SSP, many of the "modelDescription" or "SystemStructure" fields are not defined, and the user is not provided with the choice to specify them. A simple example is the author field – that can be reached using PyFMI by "model.get_author()", where "model" represents the loaded FMU. For LOTAR purposes, the author's name is highly relevant, so are many other missing fields. It would be beneficial for the tool vendors to provide support for the missing fields.
- [Major] Improve support of SSP. While the FMI standard is highly supported by many tool vendors, the SSP standard lacks application support – only a few tools include the option. Several use cases (Thomas 2015) would benefit from wider SSP deployment. Long time archiving and retrieval of structured system models would offer additional options for the archivist.
- [Minor] Add a documentation bridge. Modelica is seen by the LOTAR MBSE workgroup as one of the main languages for model development. The Modelica specification includes a documentation-annotation that enables embedding the model description and experimental frame. Nevertheless, the effort the model developer expends when creating a Modelica model is lost when exporting the model as an FMU or SSP. The MoSSEC or MIC information would be extremely valuable. A path to explore would be the recommended approach to "compile" the documentation as HTML – similar to how it is done in libraries – in the resource folder of the FMU or SSP. Exporting only the top-level documentation would be appreciated by the model consumers.
- [Minor] Contribute to the reference results format specification. As discussed in the self-criticism section, it is relevant to specify how an FMU or SSP should be tested to validate its behavior and verify its integrity. This seems as relevant for the LOTAR purposes as it should be for the FMI and SSP standards.

DOI 10.3384/ecp2118149

- [Suggestion] Better support of metadata in the Modelica language. Allow the specification of many of the metadata (e.g. LOTAR fields) in a structured form in the Modelica model itself, maybe by embedding such a manifest. Then the manifest can be moved automatically from the source, to the FMU, and parts extracted to the SSP if needed. This would prevent changing the FMU after its generation, to add the manifest in the extra folder.

This section should act as a trigger for discussion or call for cooperation on these topics. The LOTAR MBSE workgroup would welcome any further joint actions with the Modelica Association and its members.

4.3 Further discussion

Several additional points are currently under discussion in the workgroup. Two are discussed here:

- In which form the model shall be archived? In the first implementation of the archiving process, Modelon included the Modelica source code as a resource in the FMU. This brings advantages for a future use of the model after retrieval. Nevertheless, what format of the model shall be stored in the archive is yet to be defined as this shall be generic to any software and prevent model corruption in the future.
- How can we ensure the framework to simulate the model will be available in the future? There are many dependencies for a model simulation: a compatible operating system, python packages, etc. Current discussions involve, for example, a "dynamic" archiving platform – that could perform regression tests of the stored at each dependency update – or to store an image/container of the dependencies together with the model. There seems to be a trade-of between heavy platform implementation and heavy archive files.

Validation and Verification of the LOTAR is another highly relevant point, which could have its own paper.

5 Summary and conclusions

The LOTAR MBSE workgroup aims at standardizing the long-term usage of models – driven by the aerospace industry's needs. The archiving process would also be applicable and valuable to other industries. A proper archiving and retrieval process would ensure model capitalization and reusability. Modelica is seen as one of the main languages for future model development, and the FMI and SSP standards provide a solid foundation for the EN/NAS 9300 Part 520 standard.

A prototype implementation of the process described in the Part 520 was conducted in both a tool agnostic way and within Modelon Impact – as a tool-vendor proof of concept. This work proved the suitability of the process and confirmed the LOTAR MBSE workgroup's recommendation of relying on FMI and SSP standards. This work also enabled identifying the next lines of actions on both the development of the Part 520 and the standards used – especially FMI and SSP. Recommendations addressed in this paper are from the perspective of any general user who may need to replicate this work. The LOTAR MBSE workgroup would welcome any further joint actions on the identified items.

References

Coïc C., Andreasson J., Pitchaikani A., Åkesson J. and Sattenapalli H., (2020). "Collaborative Development and Simulation of an Aircraft Hydraulic Actuator Model". Presentation: *Asian Modelica Conference*, Tokyo, Japan.

Coïc C., Hübel M. and Thorade M., (2020). "Enhanced Steady-State in Modelon Jet Propulsion Library, an Enabler for Industrial Design Workflows". Proceedings of the American Modelica Conference 2020, Boulder, Colorado, USA, March 23-25, 2020.

INCOSE Technical Operations, (2007). *Systems Engineering Vision 2020, Version 2.03*. International Council on Systems Engineering, San Diego, CA, USA. Technical Publication: INCOSE-TP-2004-004-02.

LOTAR International, (2021). "LOTAR Standard, Overview on Parts". URL: https://lotar-international.org/lotar-standard/

Modelon, 2020. Modelon Impact "*Lowering barriers and bridging gaps*". URL: https://www.modelon.com/modelon-impact-introduction/

Modelica Association, (2021). *Modelica – A Unified ObjectOriented Language for Systems Modeling. Language Specification Version 3.5, Revision 1*. Tech. rep. Linköping: Modelica Association. URL: https://www.modelica.org/documents/MLS.pdf [MOD21]

Modelica Association, (2020). *Functional Mock-up Interface, Standard specification*, Version 2.02. URL: https://github.com/modelica/fmi-standard/releases/download/v2.0.2/FMI-Specification-2.0.2.pdf

MoSSEC, (2021). "*Modelling and Simulation information in a collaborative Systems Engineering Context, Developer's Overview*". Data Standard: ISO 10303-243. URL: http://www.mossec.org/

PyFMI, (2020). "*Python package for loading and interacting with Functional Mock-Up Units"*, Branch Version 2.6.x. URL: https://github.com/modelon-community/PyFMI

Thomas, E., Thomas, O., Bianconi, R., Crespo, M. and Daumas J., (2015). "*Towards Enhanced Process and Tools for Aircraft Systems Assessments during very Early Design Phase"*. Proceedings of the 11th International Modelica Conference, Versailles, France.

Nallon J. & Williams M., (2020). "MBSE Tools Database Update, and Integrate Models with Tools". Presentation: INCOSE International Workshop, TIMLM Working Group, Torrance, CA, USA. URL: https://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:incose_mbse_iw_2020:iw2020_timlm_mbseworkshop.pdf

W3C, (2006). "Extensible Markup Language (XML)", Version

1.1 (Second Edition), World Wide Web Data Standard. URL: https://www.w3.org/TR/2006/REC-xml11-20060816/

Williams M., Mendo J. and Nallon J., (2021). "*Where is your Roadmap for implementing MBSE Data Standards?*" Presentation: INCOSE International Workshop, TIMLM Working Group, Torrance, CA, USA. URL: https://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:incose_mbse_iw_2021:iw2021_mbse-standards_timlm.pdf

Python Software Foundation, (2021). "*Working with zip archives*". The Python Standard Library, Data Compression and Archiving, documentation library. URL: https://docs.python.org/3/library/zipfile.html

DOI 10.3384/ecp2118149