Bridging Requirements and Design in MBSE: Leveraging SysML v2 for Automatic Verification

Filippo Mazzoni
Department of Mechanical and Aerospace Engineering - DIMEAS
Politecnico di Torino
Torino, Italy
filippo.mazzoni@polito.it
ORCID: 0000-0001-6674-7113

Eugenio Brusa
Department of Mechanical and Aerospace Engineering - DIMEAS
Politecnico di Torino
Torino, Italy
eugenio.brusa@polito.it
ORCID: 0000-0001-7478-0650

Cristiana Delprete
Department of Mechanical and Aerospace Engineering - DIMEAS
Politecnico di Torino
Torino, Italy
cristiana.delprete@polito.it
ORCID: 0000-0002-0220-4707

Camilo Andrés Manrique-Escobar
Leonardo Innovation Labs & IP
Leonardo S.p.A.
Torino, Italy
camiloandres.manriqueescobar.ext@leonardo.com
ORCID: 0000-0002-9917-0215

Alberto Dagna
Digital Solutions & Engineering
Leonardo S.p.A.
Torino, Italy
alberto.dagna@leonardo.com
ORCID: 0000-0003-0407-6283

Grazia Accardo
Leonardo Innovation Labs & IP
Leonardo S.p.A.
Torino, Italy
grazia.accardo@leonardo.com
ORCID: 0000-0003-1435-0135

***Abstract*—Requirement analysis and verification are paramount in Model-Based Systems Engineering (MBSE), particularly in the aerospace industry, where safety-critical systems must be verified from the early design steps. An innovative approach leveraging the Systems Modelling Language (SysML v2) and a hydrogen tank sizing model are presented here, enabling a direct link between requirements and design. Interoperability between models of different natures is enhanced, and the automatic verification of a set of requirements is proposed, addressing the limitations imposed by SysML v1. The solution developed utilises a textual syntax, making it accessible to non-experts in this language and addressing the industry's needs, which require many engineers to write requirements. The methodology presented herein outlines the basic rules of writing requirements, their translation into SysML v2 syntax, their conversion into various formats, including tables and Excel files, and their automatic verification. Moreover, the traceability of requirements is ensured along the thread. This work highlights the benefits of enhancing the tools' interoperability and represents the first step towards the full integration between models of different natures.**

***Index Terms*—Requirements analysis, Automatic requirements verification, SysMLv2, Tools interoperability**

I. Introduction

Verification activity is defined as the "confirmation, through the provision of objective evidence, that the requirements for a specific intended use or application have been fulfilled" (BS ISO/IEC/IEEE 15288:2023) [1]. Defining a verification universally applicable is, in practice, impossible. Performing a verification process means answering questions like: "Are we building the system right?", "Is the system what we thought?", and the steps to find answers vary depending on the application domain. Nonetheless, it remains true that requirements verification plays a crucial role in Model-Based System Engineering (MBSE), and it is recursively performed through the while design activity (Fig. 1) [2].

In recent years, several studies have investigated requirements engineering and verification from different perspectives. Ung et al. [3] focus on the formal specification and verification of an industrial software module in the heavy-vehicle industry, showing how translating 32 requirements into ASCL contracts enables the verification through the *Frama-C* framework. Dagna et al. [4] propose integrating requirements and physical models, leveraging the FRET (Formal Requirements Elicitation) tool. Pan et al. [5] developed a requirement validation tool called ValidGen, which allows for saving 40% of the time compared to traditional methods. Liu et al. present a technique and a related tool to translate high-level requirements into software verifiable and observable objects, thus ensuring that software architectures are linked with lower-level requirements and compliant with certification standards [6]. A requirements formalisation based on the Modelica requirements library is proposed by Shamai et al. [7], while Morkevicius et al. [8] introduce a new approach showing how SysML v1 models can be effectively used to support automated verification and validation of requirements. Cross-disciplinary requirements validation is improved by Hu et al. [9], who propose an ontology-centric industrial requirements validation leveraging a top-level Basic Formal Ontology (BFO) and addressing the potentials of MBSE, also enhancing the interoperability among diverse modelling languages. A systems engineering methodology based on both models and simulation was conceived by Fischer et al. [10] with the aim of minimising the development risk that can lead to failures, especially when designing complex systems. Coudert et al. [11] propose algorithms related to SysML v1 dependency graphs that determine which proofs

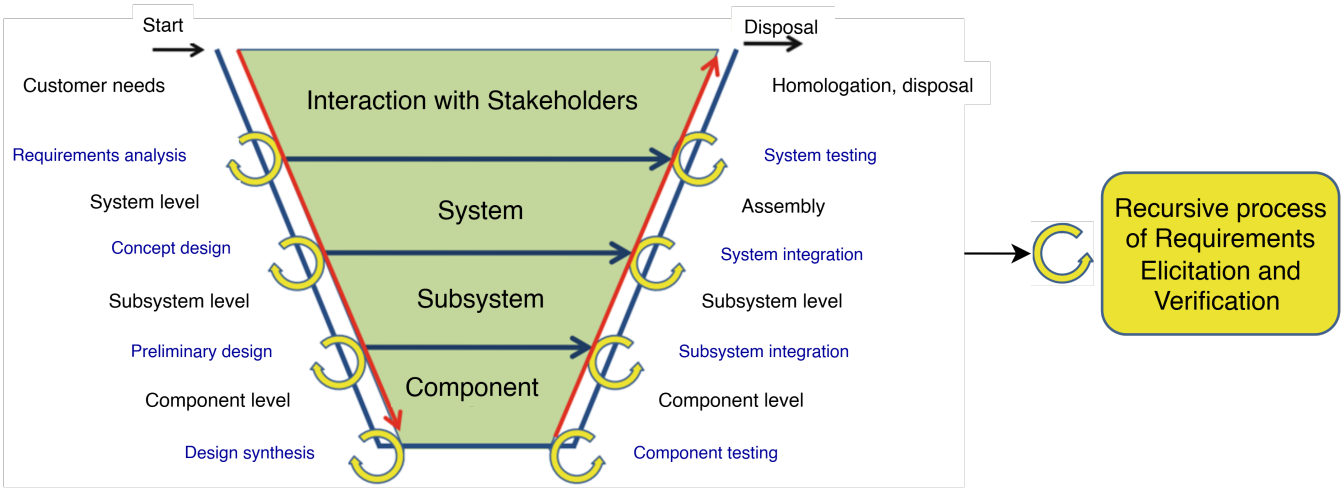

Fig. 1. Model-Based Systems Engineering "V" diagram with highlighted recurve activities (elaborated from [2].

remain valid, thereby notably reducing the time needed for re-verification activities. Agrawal et al. [12] present a web-based tool to assist in the formal verification process and ensure software correctness.

The literature review highlights a great interest in automating different tasks [13]. A study revealed that among 155 studies published between 2016 and 2022, 53% were concerned with automation in analysis and specification, and 40% with elicitation, validation, and requirements. The study also showed a prevalent use of natural language processing. However, the transition from SysML v1 to SysML v2 could represent a significant paradigm shift.

The Systems Modelling Language (SysML) is a graphical modelling language that facilitates the application of the MBSE. Together with the tools and the methodology, the language represents one of the three pillars of MBSE. In other words, by exploiting the proper tools, methods, and languages (as the SysML), the Systems Engineering methodology can be fully deployed through the MBSE.
SysML v1 is based on pure graphical notation, which makes it easy to apply and intuitive, although it can be difficult to integrate with other tools. SysML v2, being built as a coding language with a dedicated application programming interface (SysML v2 API), could bring a significant paradigm shift, drastically enhancing the interoperability and facilitating the creation of a complete digital engineering environment [2], [14]–[16].

Due to its recent introduction, publications related to SysML v2 are still limited. Studies aiming at providing a clear definition of the KerML framework are present [17]. Verification is explored in different ways and applications. For instance, a hierarchical verification approach of analog/mixed-signal, holding the advantage of detecting inconsistencies and over-specification from the early design stage, is presented [18]. Other studies propose a model-driven verification framework for ensuring functional correctness in avionics systems as well as an approach for analysing SysMLv2 models through constraint propagation [19], [20]. Overall, SysML v2 interoperability appears to be the major trending topic. It is discussed and applied in various studies in the literature [21]. An end-to-end digital workflow incorporating an MBSE model that includes requirements and architecture modelled with a SysML v2 editor has been created for a digital constellation system [22]. SysML v2 has also been applied to the development of safety-critical avionics systems to address current difficulties in the interoperability of actual models, leveraging the SysML v2 API [23]. Another study presents the results of integrating SysML v2 and the OpenModelica open-source tool, specifically the Common Requirement Modelling Language (CRML), a formal language designed for expressing requirements in a multidisciplinary context [24]. The integration between SysML v2 and LLMs to enhance SE is also investigated, revealing the potential for reducing design times and human errors [25].

II. Industrial need & scope

Very complex systems and stringent standards are typical of the aviation technical domain. Aircraft are systems composed of several sub-systems, which can be further decomposed into many other sub-systems and components. All these elements should be compliant with specific standards and perfectly integrated to cooperate and achieve common purposes. The design should consider the entire system life cycle from different perspectives, and it should also reduce and manage complexity to decrease the risk of failure and realise a successful system. Such a challenging scenario is particularly suitable for the application of MBSE. However, to ensure a seamless integration among all the systems, the interoperability among different software must be assured. In satisfying the need to apply MBSE, the industry is hindered

by poor software interoperability. The current study aims to address part of this challenge, introducing a preliminary version of an automatic requirements verification tool. Indeed, as highlighted in Figure 1, requirements are elicited and recursively verified at each design step in MBSE. The system design can encompass thousands of requirements, leading to extremely time-consuming verification processes, particularly in the case of innovative systems, where the design space might be somehow unclear from the outset and multiple design iterations are required. It follows that the verification process should be made as standard and automatic as possible.

Thus, the scope of the work proposed herein is to provide the first version of a tool that enables the automatic verification of requirements and is easily embeddable with other tools, thereby reducing the time needed for verification. In the present study, the interoperability provided by SysML v2 and its dedicated API is leveraged to bridge the gap between requirements and design. The digital thread designed for the automatic verification of requirements is explained and applied to the case of a hydrogen tank. The digital toolchain developed aims to address a relevant industrial need.

III. Methodology

Addressing such industrial needs can be challenging, and developing tools that enable rigorous and agile design processes is mandatory. Considering that requirements writing, updating, or auditing can occur at any time without the supervision of an expert, the requirements themselves must be simple and easy to understand and write. The rules and guidelines for properly writing and managing requirements are beyond the scope of this work. However, the principle followed is that the requirements must be SMART (Specific, Measurable, Achievable, Relevant, and Traceable) [2]. Particular attention is paid to the measurability of requirements. Practically speaking, for this preliminary version of the tool, each requirement is assumed to be measurable with a single metric, and its verification must be satisfiable with a "True/False" condition.

Another fundamental aspect is the correct use of a commonly shared nomenclature used to name the design parameters. Once requirements are well written, they are verified by comparing numerical values and variables coming from the sizing model and the requirements, respectively. Correct verification through the life-cycle stages is practically unfeasible if the same variable or parameter is named differently during its multiple usages. Using a single source of truth, a proper package structure where all variables are stored and classified, glossaries, and applying reference standards are fundamental preconditions to ensure the correct functioning of digital tools [15].

A high-level view of the five elements constituting the digital thread for automatic requirements verification is shown in Figure 2, where the orange component is coded in SysML v2 language while the blue ones in Python. Starting from the left side of the image, the "SysML v2 model" is the block of code where the requirements are written in that language. In this particular case, the code has been written manually, but there are tools under development that will be capable of translating the traditional graphical notation into SysML v2 language. The SyML v2 API is used to enable communication between the orchestrator and the local server and between the SysML v2 model and the local server. In practice, once the SysML v2 model is ready, it is uploaded to the server through the API, allowing the Python orchestrator, which interacts with the local server via the API, to retrieve all the information from the SysML v2 model, such as the requirements text and hierarchy. On the right side of the figure is the sizing model of the test case, being the hydrogen tank, written in Python, that is directly linked to the orchestrator that can provide inputs and retrieve outputs for the hydrogen tank sizing. In practice, interacting with the orchestrator, the user can retrieve information from the SysML v2 model and perform design calculations through the sizing model. The activities followed to perform the automatic verification of the requirements are reported in Figure 3, where, according to the same colour code previously adopted, the steps related to SysML v2 modelling are in orange, while those related to hydrogen tank sizing modelling are in blue.

Assuming that the SysML v2 model has already been uploaded into the local server, the verification starts with a command sent by the systems engineer. The Python orchestrator sends a pre-compiled query to the SysML v2 API that retrieves all the data related to requirements from the local server and sends it to the Python orchestrator. The orchestrator parses the data and converts it into a human-readable format. Then, the hydrogen tank sizing model is imported, the values of the input parameters are identified among the requirements data and set as sizing inputs to the model that is run. The numerical outputs are then stored as an array. Moving back to SysML v2 related activities, the constraining conditions imposed by the requirements are collected and stored in another array. These two arrays are then compared and, as a result, a true or false output is obtained for each condition, which is then converted into a "Passed" or "Failed" result when the verification table is generated. Finally, the systems engineer can analyse the table and evaluate the outputs of the automatic verification process. It is worth highlighting that applying a correct and shared naming convention is crucial for performing the verification, because if the name given to the same variable contained in the model and in the requirements is not identical, the automatic verification cannot be executed.

IV. Application & Discussion

The digital toolchain described in the previous section is applied to the hydrogen storage vessel to test its reliability and prove its functionality. Hydrogen storage systems are very complex and critical from a safety point of view, and their design can be very challenging from the very early design stages [26] [27]. The high complexity and innovation level characterising such a system typically create many requirements and multiple design optimisations, even in the

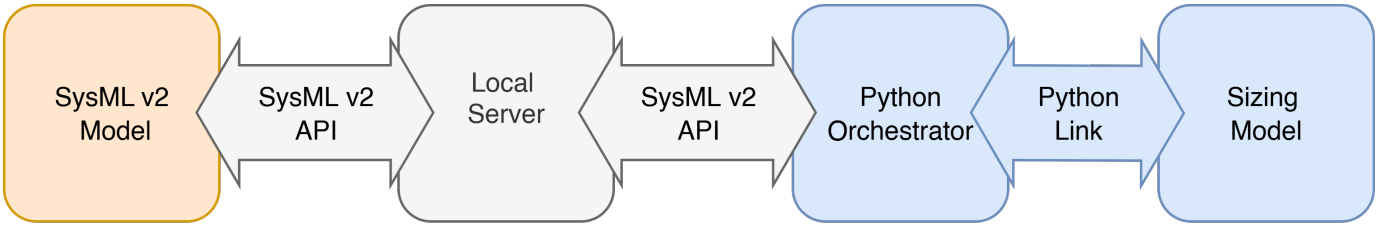

Fig. 2. High-level components constituting the automatic verification digital thread: SysML v2 model (in orange), Python models and orchestrator (in blue), and the local server communicating through the SysML v2 API (in grey).

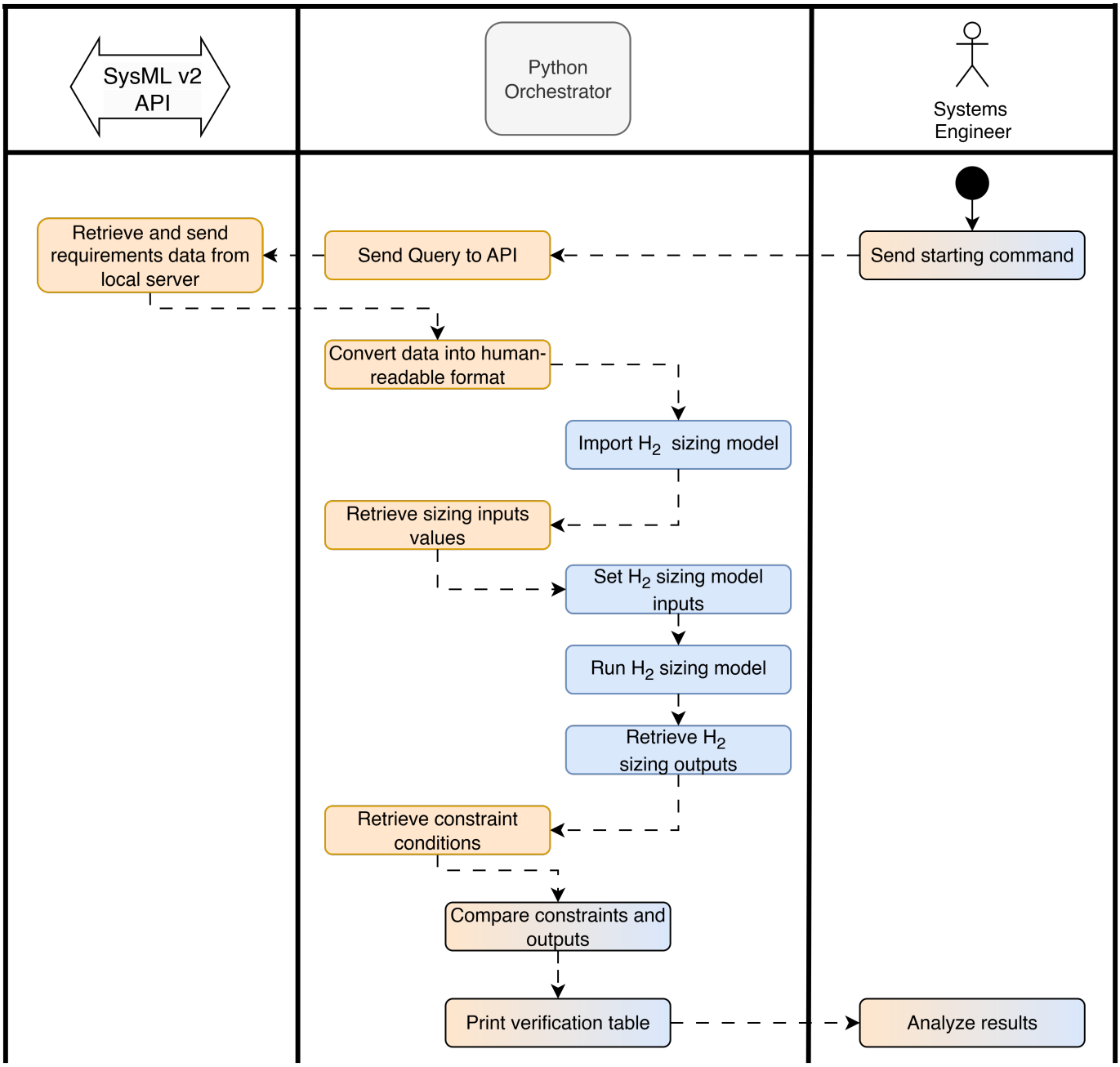

Fig. 3. Sequence diagram of the Python orchestrator (activities related to SysML v2 are in orange and to model sizing in blue).

preliminary phase. Automatic requirements verification would be suitable in this context to save time, allowing engineers to focus on more relevant tasks and automate repetitive operations.

To show the application of the previously described tool, five simple requirements have been written and reported in the diagram in Figure 4. Despite the reduced number of requirements, it is essential to observe that a simple requirements hierarchy is present. The requirements are measurable and characterised by an *attribute*, corresponding to a particular property of the hydrogen storage system, a constraining condition under the section *require constraints*, and, as usual, a name and a text. The requirements diagram shown here can be created without relying on the toolchain described in Section III. However, no ID has been assigned to the requirements. The SysML v2 model used to generate the diagram is now sent to the local server through the SysML v2 API and can be retrieved from the Python orchestrator (see Figure 2).

When the SysML v2 model is uploaded on the local server, an ID is automatically assigned to each requirement. Then, the orchestrator queries the local server via the API, retrieving all the necessary information in ".json" format, parses it, and converts it into another dictionary. To allow the designer to visually inspect the required data, a table is created and exported in .csv file format, thereby enabling its visualization in commonly used software such as the Microsoft Excel (Table I). The information on the requirements hierarchy are translated into the table as a number. Indeed, the first column, named "Requirements Breakdown Structure (RBS) Number", intuitively describes the structure graphically reported in Figure 4, consisting of various father-child relationships through numerical notation.

At this point, all the information related to requirements modelling have been properly elaborated and converted. The orange action "convert data into a human-readable format" on the left side of Figure 3 has been correctly performed, and the link between the Python orchestrator and the SysML v2 model passing through the local server has been correctly implemented relying on the SysML v2 API and the robust orchestrator. Indeed, the orchestrator exhibits a high degree of flexibility and can handle different sets of requirements independently, regardless of their number.

Moving to the system of interest (SOI) sizing (blue boxes on the right side of Python orchestrator column in Figure 3), that in this case is the hydrogen tank, the link between the orchestrator and the hydrogen tank sizing model has been created through a simple import command. The sizing model was developed in accordance with the object-oriented programming principle. The imported model is a class, and the orchestrator creates an instance of it. The inputs to the model are selected from the attributes of the liquid hydrogen tank class. Then, the tank sizing is performed.

This automatic toolchain has not been conceived to be a black box. In principle, a certain degree of supervision by the user is preferable. To this purpose, at this point, the outputs of the sizing model can be viewed (Figure 5) so that the sensibility of the subject-matter expert, that is, in this case, the engineer responsible for the design of the hydrogen storage system, who can evaluate the results and determine if they are acceptable. The model's outputs are then used to verify the requirements. If the boolean condition described in the constraint is true, the requirement is "Passed"; otherwise, it is "Failed." The "print verification table", the last activity of the Python orchestrator in Figure 3, is now completed. The requirements table (Table I) has been updated with values derived from the model sizing, and the "Verification" column indicates whether the requirements are met (Table II). It should be noted that the requirements IDs are not reported in the last table (Table II) due to formatting reasons; however, they are still present in the final table.

The flexibility of this digital thread for automatic requirements verification allows us to compare multiple solutions to identify the tank configurations satisfying all the requirements, by automating the verification over different inputs and seeing which one passes the verification. Thanks to the structure of the sizing model, multiple instances of the liquid hydrogen tank class can be created, allowing for the investigation of several tanks with different characteristics with limited effort. In this application, three tanks with different storage capacities (10, 100, and 1000 kg) have been considered in the current analysis. As can be seen from Table III, the different outputs are computed for each of the three tanks, and the requirements verification has been conducted. It is worth mentioning that the time needed to perform these computations is equal to the time required to type the numerical values of the inputs, which is so fast that this process can be assumed almost instantaneous.

V. Conclusion & Next steps

This paper presents a digital toolchain that enhances requirement verification by leveraging SysML v2 and integrating it with computational models. In Section III, the architecture is described, and the steps taken to verify the requirements are outlined. Then, in Section IV, the tool has been applied to a practical example related to a liquid hydrogen tank. To focus on the potential of the digital thread presented here, the design of the hydrogen vessel has been intentionally simplified, resulting in a reduced set of five requirements to be verified and correlated to an equal number of outputs computed through the hydrogen tank sizing model.

A high degree of automation has been shown, reducing the manual efforts to verify requirements. The proposed solution can be operated by both SysML v2 and Python language experts and non-experts alike, and its usability has been demonstrated in the application (Section IV). Additionally, IDs are automatically assigned to each requirement, ensuring traceability throughout the process and minimising human errors.

There are still some limitations to be overcome. When the SysML v2 model is published on the local server, the IDs are automatically assigned to each requirement, but, for example, if a requirement is modified and a new version of the model is

TABLE I
REQUIREMENTS TABLE COLLECTING INFORMATION FROM THE SYSML V2 MODEL.

RBS Number	Name	Text	Attribute	Constraint	ID
	H2SDSysSpecifications				d1446f9a...
1	DesignConstraints				22b6a8b8...
1.1	MaximumVolume	The volume of the H2Tank shall be less than 3.0 cubic meters	outer_tank_volume	outer_tank_volume <= 3.0	c0ffb57b-2...
1.2	MaximumLength	The H2Tank shall be less than 2 meters in length	outer_length	outer_length <= 2.0	05a0ce5a-c...
1.3	MaximumDiameter	The H2Tank shall be less than 1 meters in diameter.	outer_diameter	outer_diameter <= 1.0	c20ad00b-0...
2	PerformanceRequirements				ca2e2b47-4...
2.1	GravimetricIndex	The gravimetric index of the H2Tank shall be higher than 30%.	gravimetric_index	gravimetric_index >= 30.0	528c9b5d-7...
2.2	VolumetricIndex	The volumetric index of the H2Tank shall be higher than 35%.	volumetric_index	volumetric_index >= 35.0	8945a368-88...

TABLE II
REQUIREMENTS TABLE SHOWING SIZING AND VERIFICATION OUTPUTS.

RBS Number	Name	Text	Attribute	Constraint	Value	Verification
	H2SDSysSpecifications					
1	DesignConstraints					
1.1	MaximumVolume	The volume of the H2Tank shall be less than 3.0 cubic meters	outer_tank_volume	outer_tank_volume <= 3.0	2.42	Passed
1.2	MaximumLength	The H2Tank shall be less than 2 meters in length	outer_length	outer_length <= 2.0	1.85	Passed
1.3	MaximumDiameter	The H2Tank shall be less than 1 meters in diameter.	outer_diameter	outer_diameter <= 1.0	1.22	Failed
2	PerformanceRequirements					
2.1	GravimetricIndex	The gravimetric index of the H2Tank shall be higher than 30%.	gravimetric_index	gravimetric_index >= 30.0	35.12	Passed
2.2	VolumetricIndex	The volumetric index of the H2Tank shall be higher than 35%.	volumetric_index	volumetric_index >= 35.0	60.66	Passed

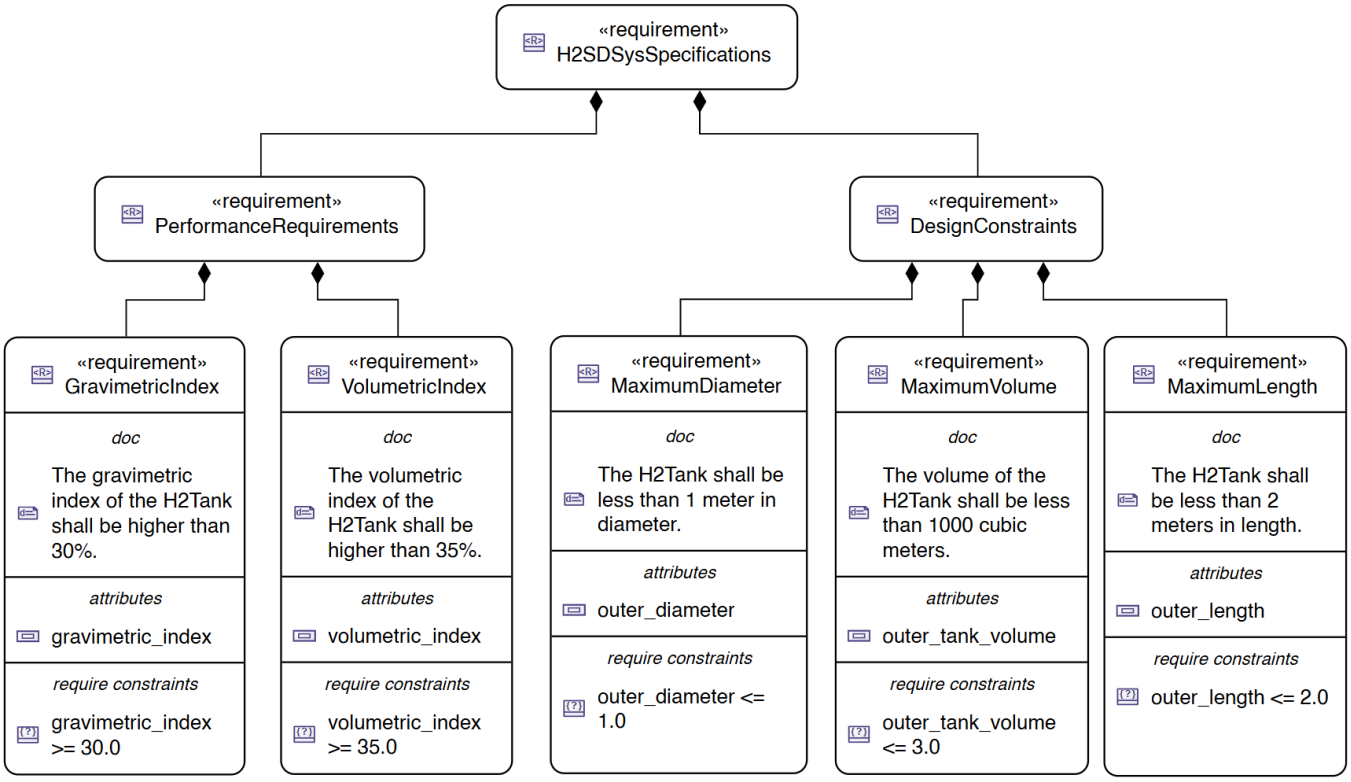

Fig. 4. Requiremetns diagram (elaborated with [28]).

TABLE III
SIZING AND VERIFICATION OUTPUTS COMPARISON BETWEEN THREE TANKS OF DIFFERENT CAPACITIES.

RBS Number	Name	H2 Tank capacity					
		10 kg		100 kg		1000 kg	
		Value	Verification	Value	Verification	Value	Verification
1	DesignConstraints						
1.1	MaximumVolume	0.4	Passed	2.42	Passed	21.79	Failed
1.2	MaximumLength	1.01	Passed	1.85	Passed	3.86	Failed
1.3	MaximumDiameter	0.67	Passed	1.22	Failed	2.55	Failed
2	PerformanceRequirements						
2.1	GravimetricIndex	16.43	Failed	35.12	Passed	49.5	Passed
2.2	VolumetricIndex	37.01	Passed	60.66	Passed	67.42	Passed

updated, then a new version is uploaded on the server, and all the IDs change. It is worth noting that, in this study, coherence among the IDs is maintained throughout the process thanks to the use of a local server. Also, in an industrial context, where interoperability must be assured and where engineers must be able to work on different parts of the same system simultaneously, this issue may become critical and should be carefully addressed. Other limitations are related to the various types of requirements, which may not be verifiable using ”Pass/Fail” criteria, and the constraints may not only be formalised as ”greater than” or ”smaller than” relations. These two latter limitations can serve as the starting point for further development of this preliminary digital toolchain. Further releases should include the possibility, for instance, of dealing with requirements constraints where the value shall be included inside a given range, and the orchestrator shall be able to specify if a requirement is not verifiable. To easily verify large sets of requirements, the following steps should also provide the number of requirements that are passed and failed. Additionally, the orchestrator should be linked to dynamic

```
Tank sizing computed for
- Al 6061-O liner material
- ['Fiberglass', 'PTFE'] insulation materials
- 100 kg of liquid hydrogen:
OUTPUTS
    - Tank volume: 2.42185373316719 m^3
    - Tank outer diameter: 1.2239386857010337 m
    - Tank outer length: 1.7708404130342916 kg
    - Tank gravimetric index: 35.1189678792133 %
    - Tank volumetric index: 60.6587457216085 %
```

Fig. 5. Sample of the hydrogen tank sizing outputs for visual inspection.

models designed with software different from Python, such as MATLAB Simulink.

One of the biggest challenges to enabling broad applicability in the industrial context is the use of a shared nomenclature that falls beyond the scope of the solution presented here, as well as standardisation in the way requirements are written. Quite often, no guidelines are provided on how to write requirements, but it is sufficient that the requirements are unique and clear. However, a standard tool to write requirements, limiting their length and structure, might be necessary to enable the following design phase by smoothly integrating an automatic requirements verification tool like the one proposed here.

In conclusion, it has been demonstrated that the digital toolchain presented here has the potential to drastically reduce the time required for requirements verification, allowing for a much quicker trade-off analysis, even in a large design space. The proposed solution is not meant to be used as a stand-alone tool but to be integrated into a much larger digital engineering environment. Even if there are still issues to be addressed, this first version of the orchestrator already shows relevant advantages, representing an essential step in enabling the connection between MBSE and design in an industrial context.

REFERENCES

[1] "BS ISO/IEC/IEEE 15288:2023: Systems and software engineering. System life cycle processes," 2023.

[2] E. Brusa, A. Calà, and D. Ferretto, *Systems Engineering and Its Application to Industrial Product Development*, ser. Studies in Systems, Decision and Control. Cham (CH): Springer International Publishing, 2018.

[3] G. Ung, J. Amilon, D. Gurov, C. Lidström, M. Nyberg, and K. Palmskog, "Post-Hoc Formal Verification of Automotive Software with Informal Requirements: An Experience Report," in *2024 IEEE 32nd International Requirements Engineering Conference (RE)*, Jun. 2024, pp. 287–298.

[4] A. Dagna, S. Centomo, E. Brusa, C. Delprete, and R. Gentile, "Automatic system requirements verification for an MBSE-oriented aircraft design process," in *ICAS PROCEEDINGS*. Florence: International Council of the Aeronautical Sciences, Sep. 2024.

[5] H. Pan and Y. Yang, "ValidGen: A Tool for Automatic Generation of Validation Scripts to Support Rapid Requirements Validation," in *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, ser. ICSE-Companion '24. New York, NY, USA: Association for Computing Machinery, May 2024, pp. 124–128.

[6] J. Liu, J. D. Backes, D. Cofer, and A. Gacek, "From Design Contracts to Component Requirements Verification," in *NASA Formal Methods*. Springer, Cham, 2016, pp. 373–387.

[7] W. Schamai, P. Helle, N. Albarello, L. Buffoni, and P. Fritzson, "Towards the Automation of Model-Based Design Verification," *INCOSE International Symposium*, vol. 26, no. 1, pp. 585–599, 2016.

[8] A. Morkevicius and N. Jankevicius, "An approach: SysML-based automated requirements verification," in *2015 IEEE International Symposium on Systems Engineering (ISSE)*, Sep. 2015, pp. 92–97.

[9] X. Hu, R. Arista, J. Lentes, J. Lu, X. Zheng, J. Sorvari, F. Ubis, and D. Kiritsis, "Ontology-centric industrial requirements validation for aircraft assembly system design," *IFAC-PapersOnLine*, vol. 55, no. 10, pp. 3016–3021, Jan. 2022.

[10] N. Fischer, "Automated validation of minimum risk model-based system designs of complex avionics systems," Thesis, Technischen Universit"at Ilmenau, 2017.

[11] S. Coudert, L. Apvrille, B. Sultan, O. Hotescu, and P. de Saqui-Sannes, "Incremental and Formal Verification of SysML Models," *SN Computer Science*, vol. 5, no. 6, pp. 1–30, Aug. 2024.

[12] A. Agrawal, E. First, Z. Kaufman, T. Reichel, S. Zhang, T. Zhou, A. Sanchez-Stern, T. Ringer, and Y. Brun, "PRoofster: Automated Formal Verification," in *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, May 2023, pp. 26–30.

[13] R. Delima, K. Mustofa, and A. K. Sari, "Automatic Requirements Engineering: Activities, Methods, Tools, and Domains – A Systematic Literature Review," *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*, vol. 7, no. 3, pp. 564–578, Jun. 2023.

[14] "Systems-Modeling/SysML-v2-Release," OMG® Systems Modeling Community, Mar. 2025.

[15] S. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide to SysML: The Systems Modeling Language*. Morgan Kaufmann, Oct. 2014.

[16] J. S. Wheaton and D. R. Herber, "Digital requirements engineering with an INCOSE-derived SysML meta-model," Oct. 2024.

[17] J. Hugues, "SysMLv2 as a DSML to support AADLv2 Semantics and Analysis," Pittsburgh, PA, 2023.

[18] S. Kwasigroch, N. Theobald, J. Koch, and C. Grimm, "A Roundtrip: From System Requirements to Circuit Variations and Back," in *DVCon Europe 2024; Design and Verification Conference and Exhibition Europe*, Oct. 2024, pp. 20–26.

[19] H. Kausch, M. Pfeiffer, D. Raco, B. Rumpe, and A. Schweiger, "Model-driven development for functional correctness of avionics systems: A verification framework for SysML specifications," *CEAS Aeronautical Journal*, pp. 1–16, Oct. 2024.

[20] A. Ratzke, S. Post, J. Koch, and C. Grimm, "Constructive Model Analysis of SysMLv2 Models by Constraint Propagation," in *2024 19th Annual System of Systems Engineering Conference (SoSE)*, Jun. 2024, pp. 239–244.

[21] M. Bajaj, S. Friedenthal, and E. Seidewitz, "Systems Modeling Language (SysML v2) Support for Digital Engineering," *INSIGHT*, vol. 25, no. 1, pp. 19–24, 2022.

[22] A. Busch, E. Vidana, and A. Luc, "Connecting MBSE to Spacecraft Modeling & Simulation," Ansys, Tech. Rep., may 2024.

[23] A. Ahlbrecht, B. Lukić, W. Zaeske, and U. Durak, "Exploring SysML v2 for Model-Based Engineering of Safety-Critical Avionics Systems," in *2024 AIAA DATC/IEEE 43rd Digital Avionics Systems Conference (DASC)*, Sep. 2024, pp. 1–8.

[24] D. Bouskela, L. Buffoni, A. Jardin, V. Molnair, A. Pop, and A. Zavada, "The Common Requirement Modeling Language," *Modelica Conferences*, pp. 497–510, Dec. 2023.

[25] J. K. DeHart, "Leveraging Large Language Models for Direct Interaction with SysML v2," *INCOSE International Symposium*, vol. 34, no. 1, pp. 2168–2185, 2024.

[26] F. Mazzoni, R. Biga, C. A. Manrique-Escobar, E. Brusa, and C. Delprete, "Design space exploration through liquid H2 tank preliminary sizing and design of experiments analysis," *International Journal of Hydrogen Energy*, vol. 95, pp. 1252–1260, Dec. 2024.

[27] F. Mazzoni, G. Accardo, R. Biga, E. Brusa, C. Delprete, C. Manrique-Escobar, and V. Vercella, "Hydrogen storage system design: case studies for airborne application," in *ICAS Proceedings*, 2024.

[28] "SysON | The NextGen SysML Modeling Tool," https://mbse-syson.org/.