AIAA SciTech Forum
12-16 January 2026, Orlando, FL
AIAA SCITECH 2026 Forum

Mokotoff, P. R., Shi, M., Carrere, A. L., and Cinar, G. (2026). Weakly Coupling MBSE and MDAO via a Tool-Neutral Authoritative Source of Truth. AIAA SciTech 2026 Forum. American Institute of Aeronautics and Astronautics. https://doi.org/10.2514/6.2026-1607

10.2514/6.2026-1607

Weakly Coupling MBSE and MDAO via a Tool-Neutral Authoritative Source of Truth

Paul R. Mokotoff*
Department of Aerospace Engineering, University of Michigan, Ann Arbor, Michigan 48109

Mingxuan Shi,†
The Boeing Company, Everett, Washington 98275

Alexander L. Carrere‡
The Boeing Company, Huntsville, Alabama 35801

Gokcin Cinar§
Department of Aerospace Engineering, University of Michigan, Ann Arbor, Michigan 48109

Digital engineering at-scale requires model-based systems engineering (MBSE) to remain traceable while multidisciplinary analysis and optimization (MDAO) evolves rapidly across distributed teams and tools. Direct tool–to–tool links create conflicting "sources of truth" because MBSE typically holds system-level requirements while MDAO environments hold subsystem- and component-level data. This paper introduces a tool-agnostic framework that resolves this conflict by establishing an authoritative source of truth (ASOT) outside the MBSE and MDAO environments. The ASOT is a version-controlled, unit- and type-aware data schema with stable identifiers. MBSE and MDAO artifacts are synchronized with idempotent read/write APIs, enabling seamless data transfer. A demonstration developed under NASA's Sustainable Flight National Partnership automatically instantiates and updates a system model from the ASOT, preserving structure, identifiers, and units. A concrete mapping between The Boeing Company's Aircraft Data Hierarchy and the SysML modeling language is enabled by open-source adapters (Jython for Magic System of Systems Architect, Python for OpenMDAO), which implement the synchronization and produce auditable change logs. The resulting weak coupling reduces training/licensing burden and supports distributed collaboration while maintaining traceability required by emerging digital engineering guidance. The framework, API code, and workflow patterns are reusable for coupling other MBSE and MDAO environments.

I. Introduction

Model-Based Systems Engineering (MBSE) has emerged as a digital means to track complex system design activities, maintain traceability between design decisions, and support requirement verification [1]. A centralized system model acts as a global repository, storing all information about the system architecture, its design requirements, and analyses performed [2], thus supporting systems design practices. A major advantage of MBSE is that a consistent set of parameters defines the system architecture – if a parameter or component is changed in one part of the model, it is modified across the entire system model. This ensures that all systems engineers use the same parameters, thus eliminating the need to manually update all design documents. However, MBSE currently lacks the capability to provide analytical and numerical methods [3], requiring connections to external analysis tools.

In contrast, Multidisciplinary Analysis and Optimization (MDAO) has been a longstanding facet of engineering design, employing computational models to simulate or optimize a system's behavior and performance. Such models are used by disciplinary engineers to evaluate a component's performance while interacting with other components in a subsystem. Furthermore, a component can be optimized for a specific objective and constraints, ensuring sufficient

*Graduate Research Assistant, Department of Aerospace Engineering, University of Michigan, Ann Arbor, Michigan 48109, AIAA Student Member.
†Propulsion Engineer, The Boeing Company, Everett, Washington 98275, AIAA Member.
‡Associate Technical Fellow, The Boeing Company, Huntsville, Alabama 35801, AIAA Member.
§Assistant Professor, Department of Aerospace Engineering, University of Michigan, Ann Arbor, Michigan 48109, AIAA Senior Member.

compatibility and performance once integrated into the overall system architecture. However, MDAO tools only perform the analyses and are not intended for rapid reconfiguration and integration into different engineering workflows [4].

Though MBSE and MDAO tools may seem disparate, their purposes are complementary. MBSE excels at representing the system architecture in an understandable manner, capturing design requirements, and easily tracing design decisions throughout the design cycle. MDAO excels at analyzing specific system architectures and identifying the optimal component design. Evidently, there is a synergy between these two tools, and combining their capabilities may help synthesize system architectures that rely on multidisciplinary teams (possibly distributed across multiple geographic locations) to design different parts of the system architecture. However, this becomes increasingly difficult as the size of the system architecture grows and more disciplinary engineers become involved [5].

A. Engineering Computational Tool Couplings in the Literature

Automated couplings between MBSE and MDAO tools have emerged in the literature, particularly as a result of the AGILE 3.0 and 4.0 projects funded by the European Union [1, 6, 7]. At first, these projects focused on integrating computational tools into MDAO frameworks, and have since been broadened to include MBSE and MDAO tool couplings. The tools are connected via a two-step process: 1) defining a data schema to facilitate information transfer; and 2) establishing (possibly automated) workflows to run analyses and pass information between the tools.

In the United States, the Air Force Research Laboratory's (AFRL's) REACT Project, initiated in 2019, aims to achieve similar goals with a stronger emphasis on coupling disparate computational tools for MDAO while also maintaining modularity to integrate MBSE tools into a design workflow [8, 9]. The majority of REACT activities have been conducted internally and disseminated primarily within AFRL and its industrial partners. Boeing and NASA participated in technical interchange meetings with AFRL regarding REACT starting in 2022, while broader details of the program became available to the open literature through a 2026 publication [8]. Separately, one paper previously published by AFRL (not under the REACT project) introduced *Philote*, a communication standard to facilitate information transfer between heterogeneous MDAO tools [10].

While existing literature demonstrating automated workflow capabilities predominantly focuses on generating mechanisms to facilitate data transfer and run computational analyses, there are two other notable works pertaining to data schema definition. The Common Parametric Aircraft Configuration Schema (CPACS) [11] was developed to establish a hierarchical definition of the aircraft and its subsystems. Although this definition is aerospace-specific, a similar data structure could be developed to represent a more general system architecture, its subsystems, and components. For MDAO workflows, the Common MDO Workflow Schema (CMDOWS) [12] defines the optimization problem, design variables, constraints, and connections between disciplines.

Prior schema development established a foundation for developing interconnected workflows between MBSE and MDAO tools. In the literature surveyed, the MBSE tool is commonly treated as a central repository, storing the data necessary for the workflow [3, 13–16]. However, other works utilize a central data schema to store the information [17, 18], with each tool accessing/updating the parameters individually. These two approaches are defined as "strong" and "weak" couplings, respectively [19], impacting the design and data transfer workflows.

1. Strong Coupling

A strong coupling centralizes the design process around MBSE, requiring both the system architecture and multidisciplinary workflows to be modeled. For example, a detailed MBSE model of an AGILE MDAO system was developed by Ciampa et al. [13]. Another approach involved constructing Requirement-Functional-Logical-Physical (RFLP) frameworks for the system architecture (a "product RFLP") and the workflow (a "process RFLP") [20].

Other workflow models were developed using MBSE's parametric or activity diagrams in conjunction with the system model [3, 15], permitting some analyses to be executed within the MBSE environment. However, it is also possible to interface external tools with MBSE, as done by Aiello et al. [14, 16]. They used Ttool [21] to extract information from a requirement diagram in MBSE and pass it to an MDAO tool.

Modeling the MBSE-MDAO workflow with a strong coupling has two key advantages. First, all product and design information resides in one, centralized location, ensuring that all engineers can access the most up-to-date data and use consistent values in all analyses, thus eliminating the need to transfer information between groups. Second, some analyses can be executed automatically using MBSE diagrams or by interfacing with external tools, further centralizing the workflow within MBSE and reducing the time to define and execute the workflow.

However, there are some weaknesses with this approach. First, the MBSE tool retains the "authoritative source of truth" (ASOT), requiring disciplinary engineers to access the latest design information by interfacing with MBSE. This contradicts the typical workflow paradigm in which MDAO stores the subsystem- and component-level requirements and parameters. Furthermore, additional costs may be incurred for training all engineers in MBSE or purchasing enough software licenses to support the entire organization. Second, the MBSE models may be product- or workflow-specific, thus limiting its generality and applicability to other product/workflow designs within an organization. This undercuts the previous advantage pertaining to the time reduction for defining and executing a workflow. Third, the MBSE model may become increasingly complex as more design disciplines are integrated into the analysis, limiting its scalability to more comprehensive design problems.

2. Weak Coupling

A weak coupling unifies the design process around a central data schema and emphasizes integrating all disciplinary models into a comprehensive analysis framework. Seamless data transfer is enabled by developing mappings between MBSE/MDAO tools and a central data schema [17]. This approach has been exemplified in automated requirement verification frameworks [18] and broader system architecture design tools [22].

There are two key strengths to this approach. First, the data schema is accessible to all engineers without requiring any MBSE software, thus cutting costs to establish more fully integrated workflows within an engineering organization. Second, multiple data schema instances can be used to represent different system architecture configurations, allowing an organization to assess multiple designs simultaneously.

Despite the strengths, there are also two weaknesses. First, multiple versions of the data schema may exist and could lead to conflicting design changes between disciplinary teams within an organization. One potential solution is to store the data schema in a version-controlled environment, facilitating a pathway to reconcile such conflicts. Second, the data schema may become prohibitively large, limiting its effectiveness to clearly and quickly communicate data about a system architecture.

B. Industry Perspectives on Engineering Tool Couplings

Current MBSE and MDAO tool couplings in industry differ from those in academia, especially within the commercial aerospace industry. Rather than centralizing design workflows around the systems engineering process, industry practices are more discipline- and tool-centric. These practices are driven by using certified tools to demonstrate product compliance under federal regulations, a major aspect of commercial airplane development. As a result, engineers use mature tools and processes to conduct certain analyses and demonstrate that the design meets the necessary regulatory requirements. Deviating from the certified tools and processes risks non-compliance, preventing airplanes from being manufactured and delivered to the customers on schedule.

Furthermore, these tools and processes contain embedded data and assumptions that have been accumulated through decades of airplane design, which prove difficult to be captured by MBSE models and processes, as well as systems engineers. Therefore, it is difficult to model the systems engineering artifacts with sufficient disciplinary fidelity to be a single source of truth.

Although this section addresses the reasons why industry currently centralizes their practices around disciplinary tools rather than systems engineering processes, the authors emphasize that integrating MBSE processes and models with certified disciplinary tools would benefit the airplane development process. Such integration is particularly important as compliance to Development Assurance Levels becomes more closely monitored by the regulators.

C. Gaps Between Industry and Academic Practices

Two key gaps were identified from the literature review and industry perspectives. First, despite the progress made by academic- and research-based entities, collaborative workflows are more difficult to implement in industry, as data is more tightly embedded within a disciplinary team's MDAO tools. While establishing fully automated and connected workflows is a conceivable and compelling problem for academics to solve, industry does not operate in such a unified manner. Second, multiple academic studies consider the MBSE tool to act as the central data repository, holding the ASOT. Thus, there is an inherent assumption that all data within an organization is easily accessible by all parties and that the design process is centralized around MBSE.

However, this approach cannot be feasibly applied in industry – the MDAO tools hold the subsystem- and component-level parameters, leading to conflicting ASOTs. Requiring organizations to centralize the design process around MBSE could pose an insurmountable challenge; the cost of restructuring internal workflows, purchasing MBSE tool licenses for each engineer, and training disciplinary engineers in MBSE may be prohibitively expensive. Furthermore, policies in defense and aerospace now explicitly call for ASOT models and data, but stop short of prescribing a single tool to fill that role, emphasizing visibility, traceability, and configuration control across the digital thread [23].

II. Objective: A Bridge to Couple MBSE and MDAO Tools

This research defines a framework for bridging MBSE and MDAO in a repeatable, tool-agnostic manner. The proposed approach adopts a centralized, version-controlled data schema as the ASOT, and utilizes read/write application programming interfaces (APIs) to map between the ASOT and each tool's native modeling language. This enables disciplinary engineers to own their MDAO artifacts while guaranteeing system-level traceability in MBSE.

The framework is intentionally weakly coupled; data is shared via the ASOT rather than direct tool-to-tool links. This reduces training and licensing costs, supports distributed teams, and generalizes the workflow across any analysis toolchain. A design workflow is demonstrated using The Boeing Company's Aircraft Data Hierarchy (ADH) as the ASOT, Magic Systems of Systems Architect (MSOSA) for MBSE, and OpenMDAO [24] for MDAO, with an open-source API [*] to automate the exchanges.

This paper reports on the design and implementation of:

1) A tool-neutral coupling, externalized by the ASOT into a version-controlled schema and accessed via idempotent read/write APIs;
2) A concrete, reusable mapping from the ADH to the SysML modeling language with unit/type preservation;
3) An open-source API implementation for MSOSA and OpenMDAO that creates models from the ASOT, updates models from modified ASOT instances, and emits auditable change logs; and
4) Operational "best practices" for distributed teams, involving stable identifiers, aliasing, and conflict resolution via version control, with minimal assumptions about tool co-location.

Together, these components provide a repeatable, weakly-coupled approach that lowers adoption cost while maintaining traceability and design governance.

The remaining sections in this paper are as follows. Section III describes the overall framework to support NASA's Sustainable Flight National Partnership (SFNP, 2021–2025). Section IV demonstrates parts of the workflow, with a fully comprehensive demonstration available online[†]. Lastly, Section V concludes the paper, summarizing key outcomes and highlighting areas of future work.

III. MBSE-MDAO Coupling Framework

This section describes the framework for coupling MBSE and MDAO tools with a centralized data schema as the ASOT. First, the interactions between MBSE/MDAO tools are described, with the ASOT acting as an intermediary between them. Second, the ASOT's structure is described, along with the aerospace-specific structure used to support NASA's SFNP. Lastly, the API routines and their interaction with the ASOT are described. Despite the specific application demonstrated, the MBSE-MDAO interactions are tool-agnostic, and can be generalized to represent communication between any two computational tools with disparate modeling languages.

A. MBSE-MDAO Interactions

The coupled MBSE and MDAO tool interactions are illustrated in Fig. 1. The three entities involved are the: 1) ASOT; 2) MBSE tool and its API; and 3) MDAO tool and its API. The framework assumes that the MBSE and MDAO tools exist in different computational environments, further generalizing the framework to span multiple organizations in (possibly) different geographic locations.

[*]`https://github.com/ideas-um/MBSAE-API`

[†]`https://github.com/ideas-um/MBSAE-API/blob/main/Demo/Demo.md`

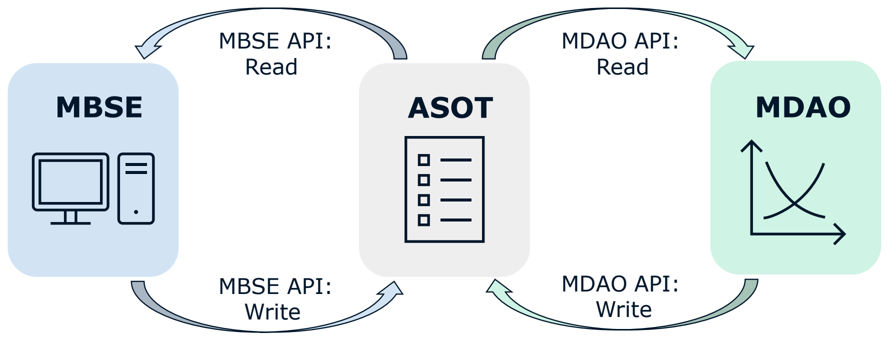


Fig. 1 MBSE-MDAO coupling framework. Boxes represent entities and arrows represent API routines.

This framework is rooted in multiple assumptions and "best practices" for design workflows:

1) **Version controlled – resolves design conflicts**: the ASOT is stored in a version-controlled repository with change history and conflict resolution capabilities, ensuring that all involved organizations use a consistent set of design parameters, requirements, and analysis results.
2) **Stable identifiers – consistent naming conventions**: every element has a stable identifier and optional aliases (for abbreviated naming conventions in computational environments).
3) **Data units/types – prevents silent data corruption**: each parameter is encoded with its own units and types.
4) **Idempotency – safe re-sync:** the API's read/write routines are idempotent, facilitating repeated synchronization.
5) **Documented design changes – ensures traceability**: changes made to the ASOT are recorded for auditing, aligning with emerging digital engineering guidance [23].
6) **Declarative mapping – consistent data transfer**: mapping rules are explicit and versioned with the ASOT.

The ASOT acts as an intermediary between the MBSE/MDAO tools and holds the data that each engineer may access. This permits systems and disciplinary engineers to each utilize their own MBSE/MDAO tools while performing their individual design tasks. All engineers may access the ASOT at any time to either: 1) receive data to update their computational models; or 2) update the ASOT with data pertaining to a design decision or analysis performed.

To perform these actions, the engineer must call an appropriate API routine, described in Section III.C. An API must be constructed for each tool, containing at least one "read" and one "write" routine (represented by the text associated with each arrow in Fig. 1). Examples of such routines are provided in Section III.D. In some cases, it is useful to have multiple read/write routines within one API. For example, in an MBSE API, a user may want one API function to create a new system model from the ASOT, while another one only updates the system model's contents.

In this work performed for NASA's SFNP, the MBSE and MDAO tools selected were MSOSA and OpenMDAO [24]. MSOSA was selected because the systems engineers at NASA and The Boeing Company had prior experience using the tool in their work and each organization held licenses to use the tool. OpenMDAO was selected because it is an open-source optimization framework co-developed by NASA.

B. ASOT Structure

The ASOT is a structured, text-based file, that acts as an intermediary between the MBSE and MDAO tools. It stores all information related to a product's design, requirements, analyses, and testing methods, and is accessed by both systems and disciplinary engineers. Each level of the ASOT contains a (sub)system or (sub)component that is nested within a higher level system or component, ensuring that parent-child relationships can be established in the MBSE/MDAO tools. Prior to synchronization with MBSE/MDAO tools, the ASOT is validated to ensure that the appropriate identifiers and data types are instantiated.

Although the ASOT is product-agnostic, an aerospace-specific implementation was used in this work – The Boeing Company's ADH [25], based on the United States Department of Defense's "Work Breakdown Structure for Defense Materiel Items" (MIL-STD-881, Revision F) [26]. The Work Breakdown Structure (WBS) meticulously and conveniently decomposes an aircraft into multiple subsystems and its parts, along with other information necessary for aircraft design development, such as system testing and evaluation.

Although the WBS provides a foundation for structuring data, additional detail is needed to categorize the information stored in each level of the structure. Data in each WBS level was further classified into four categories:

1) **Architecture:** describes the physical components utilized in the design of a subsystem.
2) **Requirements:** identifies (non-)functional design requirements.
3) **Performance:** the aircraft's performance during testing and evaluation with respect to its requirements.
4) **Behavior:** describes how the aircraft is *expected* to behave under different operating conditions.

Each component in the ADH must include a `name` (component name) and `wbs_no` (WBS number identifier), and can optionally include aliases and typed attributes organized under the *Architecture*, *Requirements*, *Performance*, and *Behavior* categories. Strings and numbers are represented as `Value Properties`, while dictionaries become `Packages` or `Blocks`, depending on the presence of a `wbs_no`. Units are stored for each parameter and preserved in the SysML model. This mapping enables repeated data synchronization while maintaining the structure prescribed by MIL-STD-881F [26]. Refer to Engelbeck et al. [27, 28] for additional information on the ADH's development and structure.

C. API Routines

An API synchronizes the ASOT and its respective MBSE/MDAO tool, as represented by the arrows in Fig. 1. Each API consists of at least one function to read the ASOT and create/update the models within its tool, and at least one function to write the changes from the tool back to the ASOT. A flowchart, illustrated in Fig. 2, further decomposes the routines into two parts: 1) reading/writing data from/to the ASOT; and 2) transforming the data to/from the MDAO/MBSE tool's modeling language.

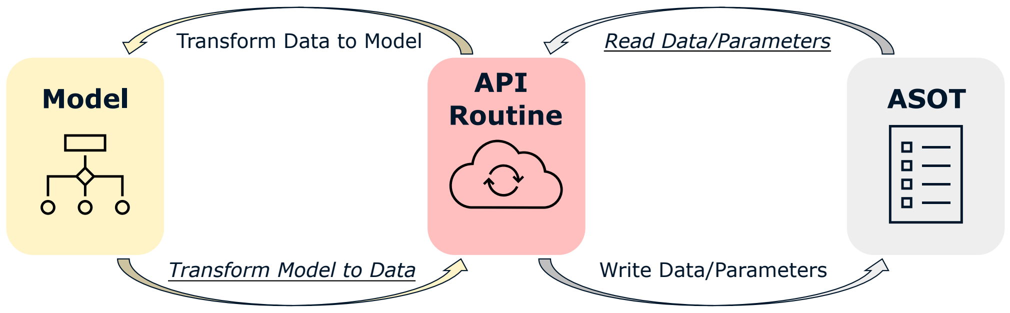


Fig. 2 MBSE/MDAO API routines. *Italic and underlined* text represents user-initialized actions.

Regardless of the action, the engineer calls the API routines within their MDAO/MBSE tool. Prior to running each routine, the engineer must specify the data that is to be read/written from/to the ASOT, indicated by the steps that are italicized and underlined in Fig. 2. This step underscores that the engineer only needs to read/write the data in the ASOT that is pertinent to the subsystem/component that they are designing. After specifying the necessary data to be transferred, the API routine runs without any human interaction. Should the engineer query the same data repeatedly, a more automated API routine could be generated to access the necessary data, further facilitating rapid data transfer between the ASOT and a given computational tool. Such automation also emphasizes the importance of devising a consistent data schema and naming convention in the ASOT.

For this work, MSOSA supports users writing "plugin" scripts to interface with external tools or files, which serve as the API between MSOSA and the ADH. These scripts are written in Jython [‡], a programming language that embeds Java routines into a Python-like syntax. The API between OpenMDAO and the ADH was written in Python [§].

D. Framework Application for NASA's SFNP

Using the framework components previously described, Fig. 3 summarizes the MBSE/MDAO tools coupled together, the ASOT selected, and the coding languages utilized for the API routines.

[‡]`https://www.jython.org/`
[§]`https://www.python.org/`

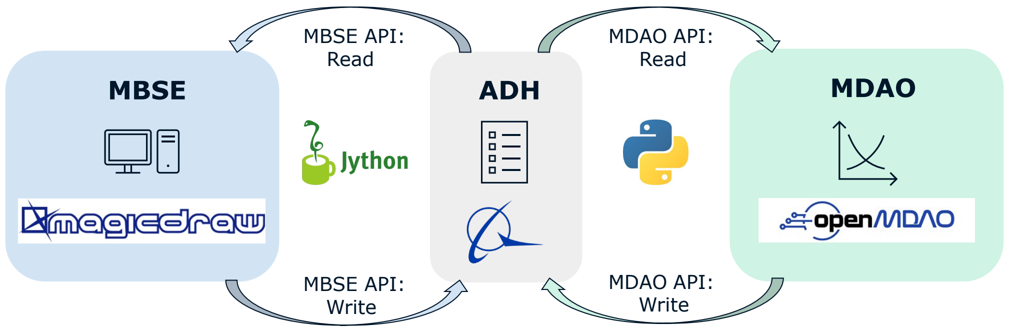


Fig. 3 MBSE-MDAO coupling framework applied to support NASA's SFNP.

To summarize the previous sections, MSOSA and OpenMDAO serve as the MBSE and MDAO tools, respectively. The structure of The Boeing Company's ADH is used in the ASOT, which acts as the intermediary between MSOSA and OpenMDAO. The API between OpenMDAO and the ADH was written using Python scripts. The API between MSOSA and the ADH was written using Jython scripts, and is open-source *. Table 1 summarizes the five routines shipped with the API, describing their directionality and typical uses.

Table 1 MBSE-ASOT API routines: direction and typical usage. All routines are idempotent; repeated calls yield the same result for an unchanged ASOT.

Routine	Direction	Effect on ADH/MSOSA	Typical Use
ReadADH	ADH → MSOSA	Create packages, blocks, value properties, and requirements from the ADH (preserving the structure, identifiers, and units).	Initialize or refresh an MSOSA model from an ADH version.
WriteADH	MSOSA → ADH	Export packages, blocks, value properties, and requirements to the ADH.	Publish MSOSA updates to the ADH for disciplinary analysis.
UpdateADH	ADH → MSOSA	Overwrite mismatched values in MSOSA from the ADH; create a change log.	Synchronize the system model with the latest ADH values.
ImportStereotypes	ADH → MSOSA	Generate stereotypes for ADH component types.	Bind domain semantics to blocks for downstream use.
WriteInstance	MSOSA → ADH	Export an Instance Specification (slots/values) to the ADH.	Capture a configuration/baseline for review or analysis.

Flowcharts of these five functions are illustrated in Figs. 7 through 11 in Appendix V.A. Along with the open-source code in the GitHub repository *, the reader may use these flowcharts to help replicate the code for their own use in other MBSE tools or tailor the existing code to their needs.

IV. Demonstration

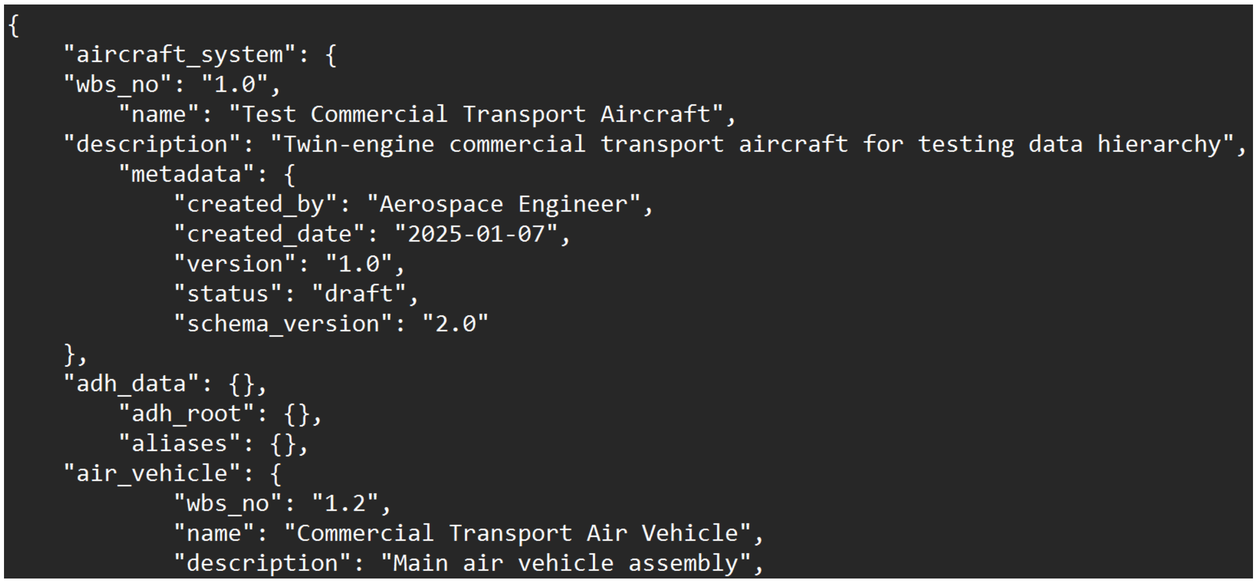
A brief demonstration illustrates how data is transferred between MSOSA and the ADH. In this paper, two steps are illustrated:

1) Importing data from the ADH to MSOSA.
2) Updating the system model in MSOSA with a modified ADH.

A more comprehensive demonstration is available online in the GitHub repository †. For the online demonstration, more specific workflow details are included such as creating an empty system model in MSOSA, inputting the ADH file names to import/export, and other details that distract from the overarching framework presented in this paper.

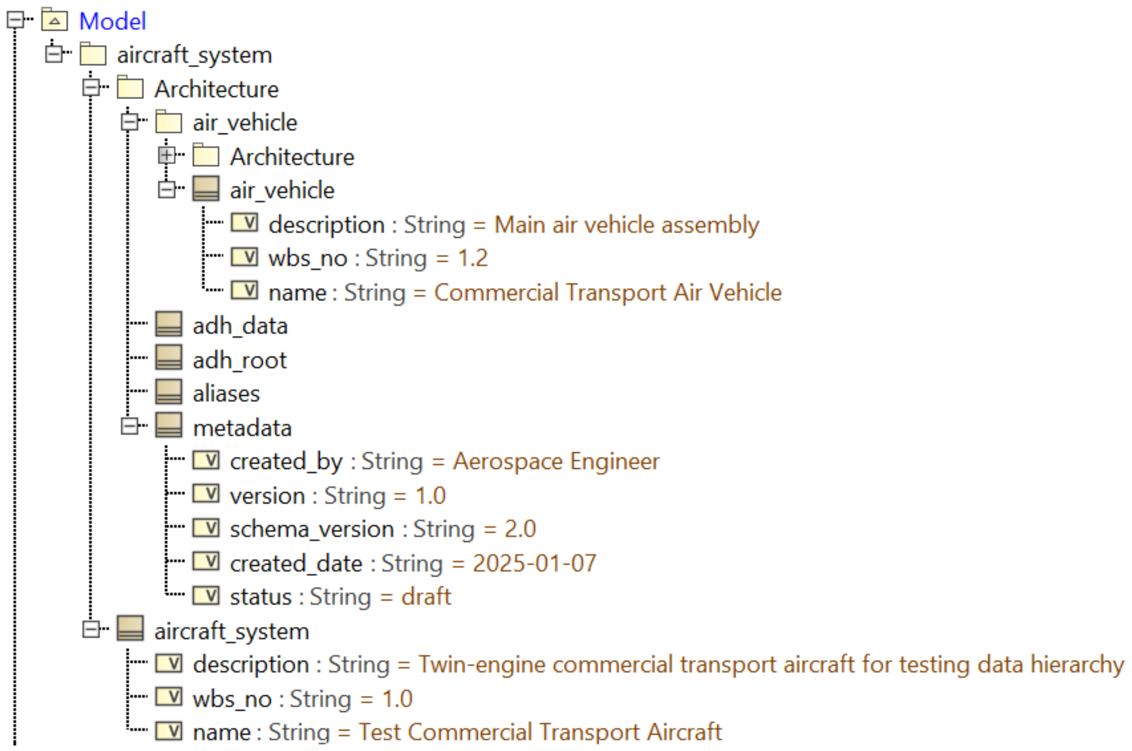
A. Importing Data from the ADH

The ADH is stored as a JSON file. Any dictionary that contains `name` and `wbs_no` key-value pairs is interpreted as a component and mapped to a SysML `Block`; its container becomes a `Package` of the same name. Keys without a `wbs_no` map to `Value Properties` (scalars retain units when provided). Arrays of components create `Packages` containing repeated `Blocks`. Under this mapping, the ADH hierarchy is preserved, and identifiers remain stable for synchronization and tracing changes. Figures 4 and 5 show the ADH snippet and the resulting SysML structure, respectively.



```
{
    "aircraft_system": {
    "wbs_no": "1.0",
        "name": "Test Commercial Transport Aircraft",
    "description": "Twin-engine commercial transport aircraft for testing data hierarchy",
        "metadata": {
            "created_by": "Aerospace Engineer",
            "created_date": "2025-01-07",
            "version": "1.0",
            "status": "draft",
            "schema_version": "2.0"
    },
    "adh_data": {},
        "adh_root": {},
        "aliases": {},
    "air_vehicle": {
            "wbs_no": "1.2",
            "name": "Commercial Transport Air Vehicle",
            "description": "Main air vehicle assembly",
```

Fig. 4 ADH excerpt from a transport aircraft configuration.



```
Model
  aircraft_system
    Architecture
      air_vehicle
        Architecture
        air_vehicle
          description : String = Main air vehicle assembly
          wbs_no : String = 1.2
          name : String = Commercial Transport Air Vehicle
      adh_data
      adh_root
      aliases
      metadata
        created_by : String = Aerospace Engineer
        version : String = 1.0
        schema_version : String = 2.0
        created_date : String = 2025-01-07
        status : String = draft
    aircraft_system
      description : String = Twin-engine commercial transport aircraft for testing data hierarchy
      wbs_no : String = 1.0
      name : String = Test Commercial Transport Aircraft
```

Fig. 5 System model created from the ADH. `Packages`, `Blocks`, and `Value Properties` are represented by folder icons, brown rectangular icons, and "V" icons, respectively.

The ReadADH function applies the mapping to instantiate the SysML model from the ADH, as illustrated in Fig. 5. The structure of the ADH is retained in the system model via nested Packages and Blocks. Inside the Block, the component's name, wbs_no, and (possibly) description are stored within it as Value Properties.

Any other data pertaining to the Block, such as subsystems/components or other data, is stored within the Architecture Package. For example, the aircraft_system has an air_vehicle as a subsystem. Since the air_vehicle also has an Architecture folder, additional subsystems and components are stored within it, but are not explored for brevity.

Aside from the subsystems, the aircraft_system has additional data stored within its Architecture Package. These entries highlight that Blocks may be made with an empty value, allowing data to be added to the ADH later. Furthermore, empty Blocks have the potential to serve as placeholders for components supplied by another organization that wishes to keep their data secret.

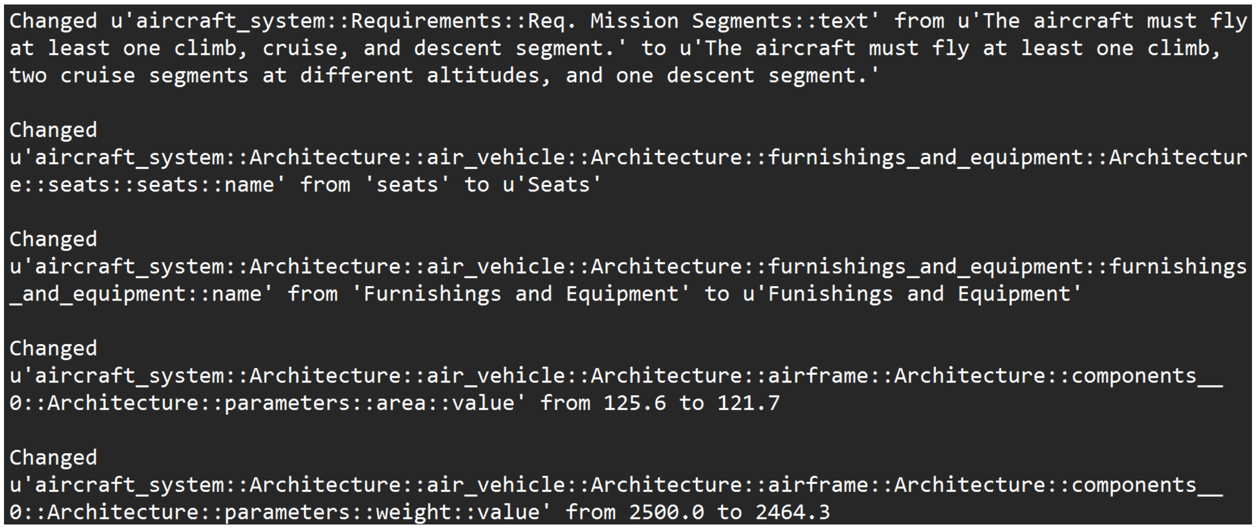
The system model within MSOSA is now accessible by a systems engineer to validate requirements, make design decisions, or modify the system architecture. If any changes are made, the systems engineer may export the MSOSA model back to the ADH, allowing disciplinary engineers to receive the updated data and, if necessary, modify their analyses.

B. System Model Updates

After the design is iterated upon, the systems engineer will need to make updates to the system model. This is achieved by using the UpdateADH function in the API, which scans the ADH and system model for any values that are different. Once identified, the corresponding values in the system model are overwritten by those stored in the ADH.

Recall that the ADH acts as the ASOT and its values take precedence over those in the system model. This is different from modifying the ADH via the system model, in which the values of the ADH are overwritten by the system model. Although the demonstration does not highlight this step, it is included in the comprehensive demonstration online [†].

Once the values in the system model are overwritten by those in the ADH, a text file is written, highlighting the changes made to the system model. An example text file is illustrated in Fig. 6. Each section of the text file highlights a change made to the system model, listing the qualified name of the model element changed and its prior/updated values.



```
Changed u'aircraft_system::Requirements::Req. Mission Segments::text' from u'The aircraft must fly
at least one climb, cruise, and descent segment.' to u'The aircraft must fly at least one climb,
two cruise segments at different altitudes, and one descent segment.'

Changed
u'aircraft_system::Architecture::air_vehicle::Architecture::furnishings_and_equipment::Architectur
e::seats::seats::name' from 'seats' to u'Seats'

Changed
u'aircraft_system::Architecture::air_vehicle::Architecture::furnishings_and_equipment::furnishings
_and_equipment::name' from 'Furnishings and Equipment' to u'Funishings and Equipment'

Changed
u'aircraft_system::Architecture::air_vehicle::Architecture::airframe::Architecture::components__
0::Architecture::parameters::area::value' from 125.6 to 121.7

Changed
u'aircraft_system::Architecture::air_vehicle::Architecture::airframe::Architecture::components__
0::Architecture::parameters::weight::value' from 2500.0 to 2464.3
```

Fig. 6 Values overwritten in the system model with updated values from the ADH.

Although the list of changes presented in Fig. 6 is rather simplistic for this example, this list may become excessively large if there are hundreds of components in the system architecture, each with its own requirements and design parameters. In operational use, the change log can be imported into a spreadsheet or dashboard for disciplinary supervisors to review. A tabular summary (model element name, prior value, new value, unit, timestamp, author) enables review at-scale and facilitates auditing within certification/verification workflows.

C. Computational Performance

Table 2 lists the number of model entities created/updated and the wall time to accomplish the synchronization tasks presented in this demonstration [¶]. The wall time was defined as the time to parse the ADH and create/update the system model in MSOSA.

Table 2 Model synchronization summary.

Metric	Model Creation (Initial ADH → MSOSA)	Model Update (Updated ADH → MSOSA)
Packages Created	44	–
Blocks Created	257	–
Value Properties Created/Updated	728	2
Requirements Parsed/Updated	5	1
Wall Time (s)	1.703	0.016

Overall, the demonstration reveals how data can be rapidly synchronized between the ADH and MSOSA while retaining its hierarchy, identifiers, units, and requirement text.

V. Conclusions

Digital engineering efforts benefit from traceable systems models while analysis environments evolve independently across tools and organizations. Conflicts arise if any one tool asserts itself as an ASOT. To overcome this challenge, this paper demonstrated an externalization of the ASOT into a version-controlled schema and synchronized design data through idempotent APIs.

The framework preserves disciplinary ownership of models and parameters while providing system-level traceability in MBSE. Stable identifiers, explicit units and types, and auditable change logs enable deterministic synchronization without mandating a monolithic toolchain. A concrete mapping from the ADH to SysML elements (blocks, packages, value properties, and requirements) makes the approach reusable beyond the specific tools shown.

A demonstration conducted under NASA's SFNP instantiated a SysML model directly from an ADH instance and then updated that model from a modified ADH while retaining its hierarchy, identifiers, units, and requirement text. The change log produced by the update routine provides a reviewable record suitable for program governance and verification workflows.

The primary implication for industry adoption is reduced licensing and training burden in MBSE with improved scalability for distributed teams. Treating the ASOT as a version-controlled artifact allows organizations to evolve analysis toolchains and collaborate across boundaries without sacrificing traceability.

One limitation of the present API is that it assumes a single writer to the ASOT and structural parity between SysML and ADH during updates. Future work includes three possible developments: 1) a `DiffADH` capability for structural and value comparison between two ASOT versions; 2) baseline management via import/export of Instance Specifications; and 3) schema-level invariants (e.g., JSON Schema or Object Constraint Language) to validate entries prior to synchronization. Public adapters for MSOSA (Jython) and OpenMDAO (Python), together with documentation and an executable demonstration, are available for adaptation to other environments.

In summary, a tool-neutral ASOT combined with disciplined synchronization closes a practical gap between MBSE and MDAO environments, enabling repeatable and auditable workflows that align with current digital engineering guidance while remaining economical to adopt.

[¶]Tests were performed on a desktop computer with a 13th Generation Intel Core i5-13400 processor (2.50 GHz) with 16 GB of installed RAM (15.7 GB usable). The desktop uses a 64-bit operating system and x64-based processor while running Windows 11 Enterprise, Version 23H2, Build 22631.5189.

Appendix

A. API Functions

As described in Section III.D, five functions comprise the API linking MSOSA and the ADH. The computational procedures for these functions are outlined using flowcharts in Figs. 7 through 11.

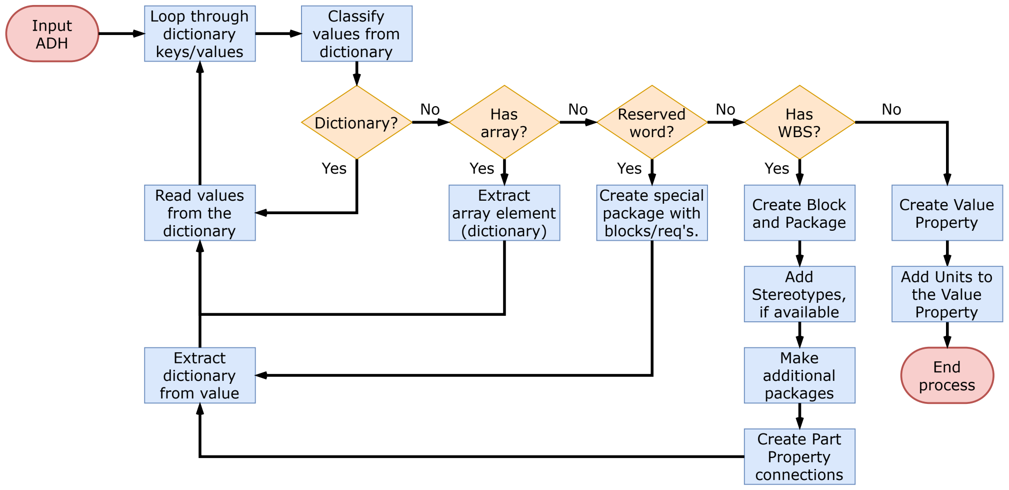

Input ADH
Loop through dictionary keys/values
Classify values from dictionary
Dictionary?
No
Has array?
No
Reserved word?
No
Has WBS?
No
Yes
Read values from the dictionary
Yes
Extract array element (dictionary)
Yes
Create special package with blocks/req's.
Yes
Create Block and Package
Create Value Property
Add Stereotypes, if available
Add Units to the Value Property
Extract dictionary from value
Make additional packages
End process
Create Part Property connections

Fig. 7 `ReadADH` **computational procedure.**

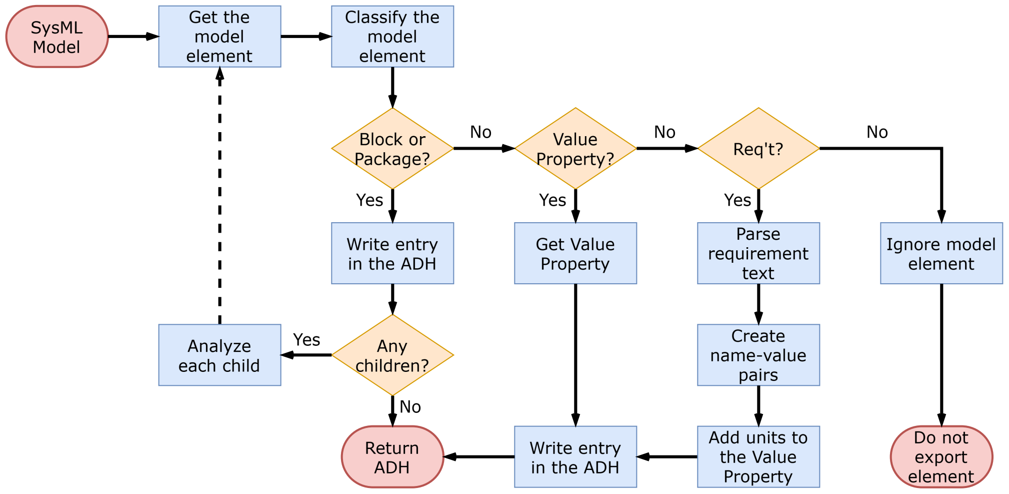


Fig. 8 `WriteADH` **computational procedure.**

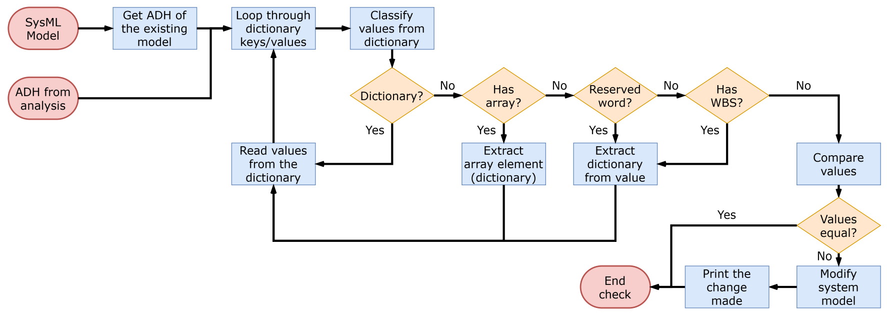

SysML Model
ADH from analysis
Get ADH of the existing model
Loop through dictionary keys/values
Classify values from dictionary
Dictionary?
No
Has array?
No
Reserved word?
No
Has WBS?
No
Yes
Yes
Yes
Yes
Read values from the dictionary
Extract array element (dictionary)
Extract dictionary from value
Compare values
Values equal?
Yes
No
End check
Print the change made
Modify system model

Fig. 9 `UpdateADH` **computational procedure.**

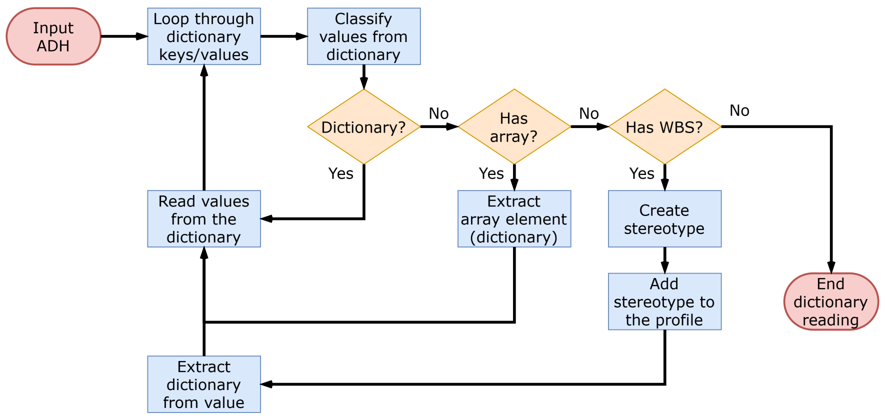

Input ADH
Loop through dictionary keys/values
Classify values from dictionary
Dictionary?
No
Has array?
No
Has WBS?
No
Yes
Yes
Yes
Read values from the dictionary
Extract array element (dictionary)
Create stereotype
Add stereotype to the profile
End dictionary reading
Extract dictionary from value

Fig. 10 `ImportStereotypes` **computational procedure.**

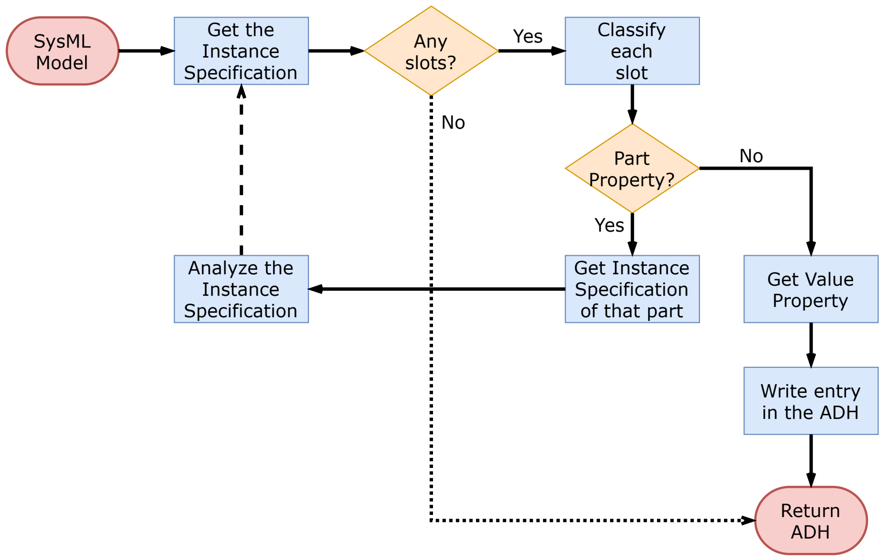


Fig. 11 `WriteInstance` **computational procedure.**

Funding Sources

This work was funded by The Boeing Company project: "Phase I Model-Based Systems Analysis and Engineering (MBSA&E) Framework Development & Assessment for NASA Sustainable Flight National Partnership (SFNP)". The contract number is "SSOW-BRT-L1023-0237".

Acknowledgments

The authors want to thank Bijan Fazal, Sean Wakayama, Ron Engelbeck, Joerg Gablonsky, and Ron Plybon for their support and valuable technical input throughout the duration of this work. The authors also acknowledge Joaquim Martins, Safa Bakhshi, and Alexandra Kerlee for their contributions to the Python-based API between OpenMDAO and the ADH.

References

[1] Bussemaker, J., Boggero, L., and Nagel, B., "The AGILE 4.0 Project: MBSE to Support Cyber-Physical Collaborative Aircraft Development," *INCOSE International Symposium*, Vol. 33, Wiley Online Library, 2023, pp. 163–182. https://doi.org/10.1002/iis2.13015.

[2] Delligatti, L., *SysML distilled: A brief guide to the systems modeling language*, Addison-Wesley, 2013.

[3] Habermehl, C., Höpfner, G., Berroth, J., Neumann, S., and Jacobs, G., "Optimization workflows for linking model-based systems engineering (MBSE) and multidisciplinary analysis and optimization (MDAO)," *Applied Sciences*, Vol. 12, No. 11, 2022, p. 5316. https://doi.org/10.3390/app12115316.

[4] Jeyaraj, A. K., Tabesh, N., and Liscouet-Hanke, S., "Connecting Model-based Systems Engineering and Multidisciplinary Design Analysis and Optimization for Aircraft Systems Architecting," *AIAA AVIATION 2021 FORUM*, 2021, p. 3077. https://doi.org/10.2514/6.2021-3077.

[5] van Gent, I., and La Rocca, G., "Formulation and integration of MDAO systems for collaborative design: A graph-based methodological approach," *Aerospace Science and Technology*, Vol. 90, 2019, pp. 410–433. https://doi.org/10.1016/j.ast.2019.04.039.

[6] Ciampa, P. D., Prakasha, P. S., Torrigiani, F., Walther, J.-N., Lefebvre, T., Bartoli, N., Timmermans, H., Della Vecchia, P., Stingo, L., Rajpal, D., et al., "Streamlining cross-organizational aircraft development: results from the AGILE project," *AIAA Aviation 2019 Forum*, 2019, p. 3454. https://doi.org/10.2514/6.2019-3454.

[7] Ciampa, P. D., and Nagel, B., "Accelerating the Development of Complex Systems in Aeronautics via MBSE and MDAO: a Roadmap to Agility," *AIAA Aviation 2021 Forum*, 2021, p. 3056. https://doi.org/10.2514/6.2021-3056.

[8] Lupp, C. A., Kao, J. Y., Novotny, N., Wontor, T., Xu, A., Singleton, J., and Sandler, D. A., "A Theoretical Foundation and Practical Demonstration of Integrating MDO, MBSE and the Digital Thread," *AIAA SciTech Forum*, 2026.

[9] Kolonay, R. M., "MDAO - An Air Force Perspective," , 2021. URL https://www.phoenix-int.com/wp-content/uploads/2021/01/Kolonay_PIMDAOEvent2021Rev3.pdf.

[10] Lupp, C. A., and Xu, A., "Creating a Universal Communication Standard to Enable Heterogeneous Multidisciplinary Design Optimization," *AIAA SCITECH Forum*, 2024, p. 1799. https://doi.org/10.2514/6.2024-1799.

[11] Alder, M., Moerland, E., Jepsen, J., and Nagel, B., "Recent advances in establishing a common language for aircraft design with CPACS," , 2020. URL https://www.researchgate.net/profile/Marko-Alder-2/publication/344346475_Recent_Advances_in_Establishing_a_Common_Language_for_Aircraft_Design_with_CPACS/links/678184023e33dd0be9f7264c/Recent-Advances-in-Establishing-a-Common-Language-for-Aircraft-Design-with-CPACS.pdf.

[12] van Gent, I., La Rocca, G., and Hoogreef, M. F., "CMDOWS: a proposed new standard to store and exchange MDO systems," *CEAS Aeronautical Journal*, Vol. 9, No. 4, 2018, pp. 607–627. https://doi.org/10.1007/s13272-018-0307-2.

[13] Ciampa, P. D., La Rocca, G., and Nagel, B., "A MBSE approach to MDAO systems for the development of complex products," *AIAA Aviation 2020 Forum*, 2020, p. 3150. https://doi.org/10.2514/6.2020-3150.

[14] Aïello, O., Kandel, D. S. D. R., Chaudemar, J.-C., Poitou, O., and de Saqui-Sannes, P., "Populating MBSE models from MDAO analysis," *2021 IEEE International Symposium on Systems Engineering (ISSE)*, IEEE, 2021, pp. 1–8. https://doi.org/10.1109/ISSE51541.2021.9582519.

[15] Hossain, N. U. I., Lutfi, M., Ahmed, I., Akundi, A., and Cobb, D., "Modeling and analysis of unmanned aerial vehicle system leveraging systems modeling language (SysML)," *Systems*, Vol. 10, No. 6, 2022, p. 264. https://doi.org/10.3390/systems10060264.

[16] Aïello, O., Poitou, O., Chaudemar, J.-C., and De Saqui-Sannes, P., "Sizing a Drone Battery by coupling MBSE and MDAO," *11th European Congress Embedded Real Time Systems (ERTS) 2022*, 2022, p. 03740634.

[17] Bussemaker, J., Boggero, L., and Ciampa, P. D., "From system architecting to system design and optimization: A link between MBSE and MDAO," *INCOSE International Symposium*, Vol. 32, Wiley Online Library, 2022, pp. 343–359. https://doi.org/10.1002/iis2.12935.

[18] Bruggeman, A.-L. M., and La Rocca, G., "From requirements to product: An MBSE approach for the digitalization of the aircraft design process," *INCOSE International Symposium*, Vol. 33, Wiley Online Library, 2023, pp. 1688–1706. https://doi.org/10.1002/iis2.13107.

[19] Chaudemar, J.-C., and de Saqui-Sannes, P., "MBSE and MDAO for early validation of design decisions: a bibliography survey," *2021 IEEE International Systems Conference (SysCon)*, IEEE, 2021, pp. 1–8. https://doi.org/10.1109/SysCon48628.2021.9447140.

[20] Swaminathan, R., Sarojini, D., and Hwang, J. T., "Integrating mbse and mdo through an extended requirements-functional-logical-physical (rflp) framework," *AIAA AVIATION 2023 Forum*, 2023, p. 3908. https://doi.org/10.2514/6.2023-3908.

[21] Institut Mines-Télécom, "TTool," , 2016. URL https://gitlab.telecom-paris.fr/mbe-tools/TTool/.

[22] Bruggeman, A.-L., van Manen, B., van der Laan, T., van den Berg, T., and La Rocca, G., "An MBSE-based requirement verification framework to support the MDAO process," *AIAA Aviation 2022 Forum*, 2022, p. 3722. https://doi.org/10.2514/6.2022-3722.

[23] Office of the Under Secretary of Defense for Research and Engineering, "DOD Instruction 5000.97 - Digital Engineering," , 2023. URL https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodi/500097p.PDF?ver=bePIqKXaLUTK_Iu5iTNREw%3D%3D.

[24] Gray, J. S., Hwang, J. T., Martins, J. R. R. A., Moore, K. T., and Naylor, B. A., "OpenMDAO: An open-source framework for multidisciplinary design, analysis, and optimization," *Structural and Multidisciplinary Optimization*, Vol. 59, No. 4, 2019, pp. 1075–1104. https://doi.org/10.1007/s00158-019-02211-z.

[25] The Boeing Company, "Aircraft Data Hierarchy," , 2025. URL https://github.com/Boeing/aircraft-data-hierarchy.

[26] United States Department of Defense, "Work Breakdown Structures for Defense Materiel Items," , 2022. URL https://quicksearch.dla.mil/qsdocdetails.aspx?ident_number=36026.

[27] Engelbeck, R. M., Ocampo, E., Gablonsky, J., Shi, M., Wakayama, S., Carrere, A., Plybon, R., Refford, M., Mokotoff, P. R., Bakhshi, S., Kerlee, A., Cinar, G., and Martins, J., "Model-Based Systems Analysis and Engineering: Aircraft Data Hierarchy," Tech. Rep. NASA/CR-20250007045, National Aeronautics and Space Administration, 2025. https://ntrs.nasa.gov/api/citations/20250007045/downloads/CR-20250007045.pdf.

[28] Engelbeck, R., Shi, M., Gablonsky, J., Ocampo, E., Wakayama, S., and Carrere, A., "The Aircraft Data Hierarchy: A Modern Aircraft Configuration Data Standard," *AIAA SciTech Forum*, 2026.