A Method for Executing Digital System Models and Digital Twins at Scale to Enrich Fleet Health Management

Daniel M Newman[1]

[1] *The Boeing Company, St. Louis, Missouri, 63134, United States of America*
daniel.m.newman@boeing.com

ABSTRACT

The digitization of the aviation industry brings the promise of increased transparency into the engineering design across aircraft lifecycles. With an increased focus on Model-Based Engineering, design engineers are constructing interoperable, standardized simulation models which can be leveraged across the Digital Thread. This advancement presents an opportunity to improve health management and prognostics by directly comparing these models against in-service data. This paper proposes a method for leveraging engineering Digital System Models and Digital Twins at-scale to facilitate and enrich health management activities. Supporting considerations include specifying model standards and ingesting fleet-level data using cloud computing.

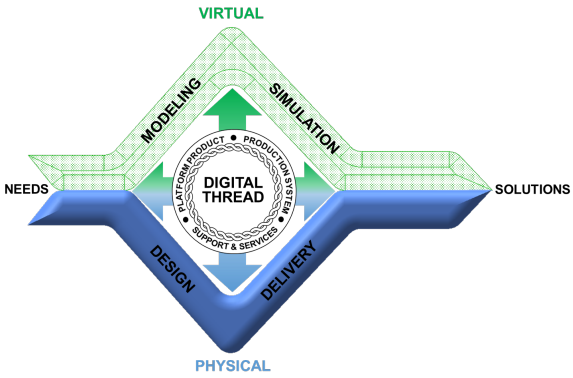

Figure 1. Model Based Engineering Diamond

1. INTRODUCTION

The transformation of the aviation industry into a digital-based engineering paradigm brings the promise of increased access to design data across the product lifecycle (Hatakeyama, Seal, Farr, & Haase, 2018). This improved transparency will be foundational to more efficient products from design to manufacture, delivery, and support. Two key technological concepts underpin this transformation: the Digital Twin and the Digital Thread. As an aspirational goal, the Digital Twin is a perfect digital replica of a specific asset, containing all of the relevant information to support design, production, delivery, operations and maintenance throughout its lifecycle. This is enabled by the Digital Thread, which connects product data across its lifecycle and enables consumption of these data for a variety of uses.

The Model-Based Systems Engineering (MBSE) diamond visualizes the flow of engineering data and describes the digital artifacts generated through product development and into support and sustainment. This concept, shown in Figure 1, demonstrates how digital engineering tools are associated with physical assets throughout the design lifecycle. These virtual assets can be described as Digital Twins of their physical counterparts, linked together by the Digital Thread. Importantly, this digital connectivity applies not only from physical assets to ther digital twins, but also horizontally from design to support and sustainment. Creating these connections is critical to developing improved descriptive and prescriptive analytics.

Realizing the MBSE Diamond requires the development of an integrated system which facilitates the execution of various models and provides data interoperatbility between them (Mihai et al., 2022). This in turn mandates standardization for model development to ensure various models may be integrated into an overall MBSE architecture. This work serves as a pilot investigation into the specific methods required to implement a standardized model execution framework.

2. DIGITAL SYSTEM MODELS AND DIGITAL TWINS

In order to leverage MBSE for health management, two critical components at the heart of Model-Based Engineering: Digital System Models (DSM) and Digital Twins (DTw) must be defined. These definitions, found in Table 1, illuminate distinctions and similarities betwen the Digital Twin and Dig-

Term	Definition
Digital System Model	A calibrated digital surrogate of a product, at the level of fidelity currently available that is intended to be the enduring, authoritative source of truth for data-driven decisions
Digital Twin	A virtual representation of the properties and behaviors of a specific instance of a physical system or process that enables prediction and optimization of performance and maintains synchronization with that physical system or process through its operational life

Table 1. DSM and DTw Definitions

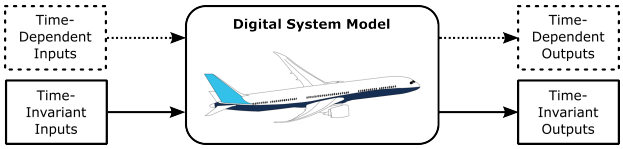

Figure 2. Baseline Digital System Model

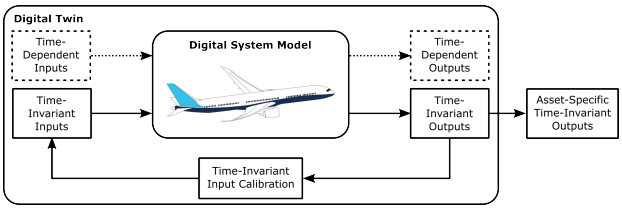

Figure 3. Baseline Digital Twin Realization

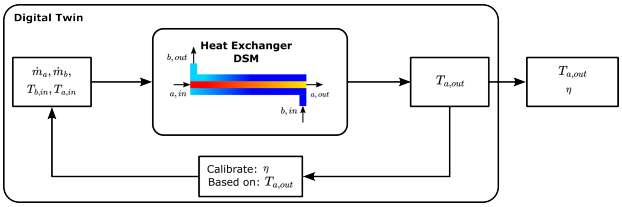

Figure 4. Baseline Digital Twin: Heat Exchanger

ital System Model. Both the Digital System Model and Digital Twin are meant to be compared with operational data to inform in-service decision-making. While the Digital System Model simulates the calibrated representation of the product, the Digital Twin simulates the behaviors of a specific instance of that product. For this reason, it is reasonable to say that the Digital Twin inherits from the Digital System Model to form the initial understanding of the specific instance under study. The DTw-specific condition/state can be realized as asset-specific operational parameters or states. While these values are represented in the underlying DSM, they are fine-tuned via the DTw as the asset goes throughout its operational life.

This work seeks to describe how a Digital System Model and Digital Twin may be evaluated against in-service data. A simple input/output interface, shown in Figure 2, summarizes this interaction. Any Digital System Model can be framed as a function which accepts some combination of Time-Dependent and Time-Invariant Inputs. The model then performs internal calculations to return a combination of Time-Dependent and Time-Invariant Outputs. On an aircraft, Time-Dependent Inputs may be populated by recorded time-series sensor data such as temperatures, pressures, altitudes, etc. By comparison, Time-Invariant Inputs may be configuration parameters (number of passenger seats, product variant type, etc.), health-related parameters (heat exchanger fouling, mechanical wear coefficient, etc.), or constant-valued sensor data (temperatures, pressures, altitudes, etc.). The Time-Dependent inputs and outputs are represented with dotted lines to indicate that not all Digital System Models make use of dynamic behavior.

Based on this understanding of the Digital System Model, a rudimentary extension of the DSM forms a representation of the Digital Twin as shown in Figure 3. This figure shows how the Digital Twin "closes the loop" between the recorded inputs and the simulated outputs to calibrate specific Time-Invariant Inputs. Critical to this feedback loop is the Time-Invariant Input Calibration step, where the simulated outputs are compared to the inputs to evaluate goodness-of-fit. Once the inputs are calibrated to a desired level of accuracy, the Digital Twin returns these asset-specific values to the user, where they may be recorded for future reference.

Defining the Digital System Model and Digital Twin in this way brings clarity to how the Digital Twin inherits and expands upon information from the underlying Digital System Model. To demonstrate this interaction, consider the basic implementation of a heat exchanger Digital Twin in Figure 4. The underlying heat exchanger DSM contains the time-invariant calculations to determine the output temperature of the hot fluid, $T_{a,out}$ based on mass flow rates, $\dot{m}_a$, $\dot{m}_b$, and temperatures $T_{b,in}$, $T_{a,in}$. Using this outlet temperature, the Digital Twin calibrates the thermodynamic efficiency, η by iteratively executing the Digital System Model. Once this calibration is complete, the Digital Twin returns the simulated outlet temperature and calibrated thermodynamic efficiency for the specific asset based on the specified inputs.

ID	A DSM Should...	Rationale
1	Be executable from a Command-Line Interface	This facilitates standardization of DSM/DTw execution
2	Be executable as a self-contained model with no external software dependencies	This ensures a DSM/DTw may be executed in any software/hardware environment, decoupled from licensed software
3	Accept parameter inputs as command line arguments and/or a path to a file containing parameter input data	This is to ensure a consistent means of providing input data
4	Return *only* the labeled output data via the command-line	This ensures standardization for the output interface of the model
5	Assume input data is in the correct engineering units	This requirement ensures no preprocessing is required to perform unit conversions for execution
6	Assume input data parameter names exactly match model variable names, including case sensitivity	This requirement ensures no preprocessing is required to rename variables prior to execution
7	Be fully executable at the designed level of fidelity based on inputs provided via the described interfaces	This requirement ensures that all required inputs are accessible via these documented interfaces

Table 2. Digital System Model Requirements

2.1. Digital System Model Interface Requirements

In order to execute Digital System Models at scale, it is necessary to define and document the interface requirements to ensure they are compatible with an execution framework as part of the Digital Thread. Defining the interface in this way, ensures scalability while enforcing minimal additional burden on the discipline engineers developing models. These requirements are provided in Table 2.

The first requirement relates to the method of execution; the DSM must be callable as an executable. This is closely related to requirement Two, which dictates that no external software dependencies may be required to execute the model. As an example, if a model is compiled as an executable named "MassSpringDamper," it must be callable like so:

```
$ ./MassSpringDamper
```

Combining these requirements essentially means that the DSM must be a self-contained, compiled executable file. This requirement exists for several reasons. First, it explicitly eliminates the need for external software licenses to execute the models. Second, it ensures parallelization using clustered computing methods available through frameworks such as Apache Spark and Apache Beam. Third, it limits the number of artifacts required to track and deploy through the model execution process. In summary, these two requirements greatly facilitate the scalability of the Digital System Models by reducing licensing cost, facilitating large-scale computation, and limiting the number of files to track.

Next, Requirements 3 and 4 describe the low-level input/output interface for the Digital System Model. In order to execute the DSM at-scale, the execution framework must be able to supply inputs in the correct format and retrieve outputs for analysis. First, the input parameters may be supplied in multiple ways via the command line: as additional arguments or as part of a data file whose path may be passed as an argument. The DSM must be capable of processing these raw inputs to populate the necessary internal input variables without any external software. Similarly, the DSM must return all relevant output variable data via the command line. These outputs must be clearly labeled so that the user may understand what values are associated with specific variables.

Using the previous example of the "MassSpringDamper" model, assume the DSM requires variables *MASS*, *SPRING_RATE*, *DAMPING_COEFFICIENT*, and *Input_Force*. To provide these variables to the model according to Requirement 3, the DSM may be invoked as follows (In a Unix environment):

```
$ ./MassSpringDamper --MASS 10. \
    --SPRING_RATE 2 \
    --DAMPING_COEFFICIENT 4 \
    --Input_Force 100.,100.,
```

The DSM then returns an output in a format like so:

```
    Velocity: 0.00,0.98,...,0.00,0.00,
    Position: 0.00,0.05,...,50.00,50.00,
```

In this example, the DSM returns a time-series output for the *Position* and *Velocity* of the mass. If the inputs were enumerated in a file named "Inputs.csv," the model may similarly be executed like so:

```
$ ./MassSpringDamper \
    --filepath Inputs.csv
```

to achieve the same output result.

Requirements 5, 6, and 7 describe integration of the Digital System Model with the Digital Thread. When executing the DSM, the formatting of the input data must be exactly correct to calculate a result at the desired level of fidelity. This begins with the units of the input data; the Digital Thread must provide inputs with the correct engineering units. Second, the names of the input parameters must exactly match the variable names used in the DSM. As the discipline engineer builds this model, they must assume both of these requirements are met upon execution. This assumption relies on accurate documentation, which will be discussed in the next section. Finally, Requirement 5 provides general guidance to the nature of DSM execution. In order for a data scientist to execute the model at the intended level of fidelity, all of the required inputs must be accessible via the command-line interface as described in Requirement 3.

3. Executing Digital System Models at-Scale

With a generalized set of interfacing requirements established, it is possible to develop a strategy for executing DSMs at-scale. This model execution framework relies on consuming Digital System Models and in-service data formatted properly according to the previous section.

The high-level workflow involved in perfoming this scalable model execution is shown in Figure 5. This process begins in the original modeling environment which the design engineers used to create the physical equations representing the system or component under consideration. Two examples — Python and Matlab/Simulink — are shown. Both environments facilitate the export and compilation of decoupled, generic executables in-line with the DSM requirements as part of this execution framework. While many other engineering design software packages exist, both Python and Matlab represent a significant portion of engineering models and are extremely flexible in their implementation and interoperatbility with other packages.

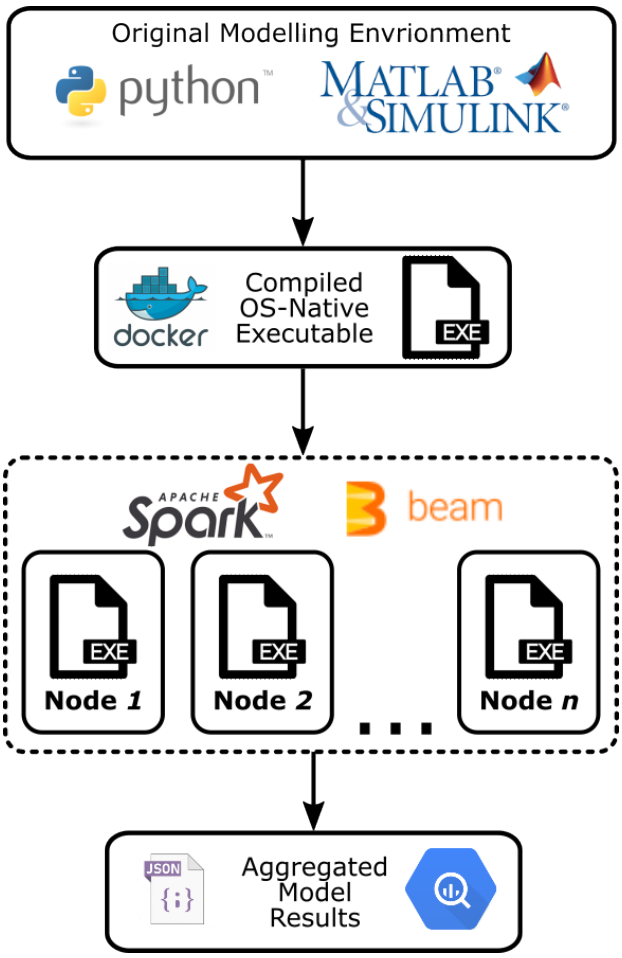

Figure 5. Model Execution Workflow

Using tools such as Docker, it is possible to cross-compile these models to execute in a desired operating system (e.g. Windows or Linux). These compiled executable files are then deployed using standard parallelization libraries such as Apache Spark and Apache Beam in conjunction with cloud and clustered computing. This step facilitates scalability bounded only by the available cloud computing infrastructure. This significantly increased throughput enables model execution at a scale consistent with the demands of aircraft fleet health management.

Lastly, this workflow uses standard JSON payloads to store the input and output of the Digital System Models. This JSON structure has the following format:

```
{"assetId": "Asset123",
 "episodeId": 123,
 "filePath": "path/to/input_data.csv",
 "argument1": 1,
 "argument2": 2,
 "argument3": 3}
```

where *assetId* identifies the specific asset associated with the data, *episodeId* identifies the instance, such as specific flight, from which the data originates, *filePath* identifies any csv-formatted input data, and the *arguments* contain parametric data relevant to the current episode. This format relies on the input-output interface defined in Section 2.1 and enables parallelizeable execution.

After the parallelized, scaled execution is completed, these outputs are stored in the cloud using the data aggregation method of choice for data science activities. For example, these JSON payloads may be stored in a Google BigQuery database for easy access.

In summary, using the DSM interface requirements as a baseline, this work proposes a workflow to execute large numbers of models at scale. Using parallel computing and efficient,

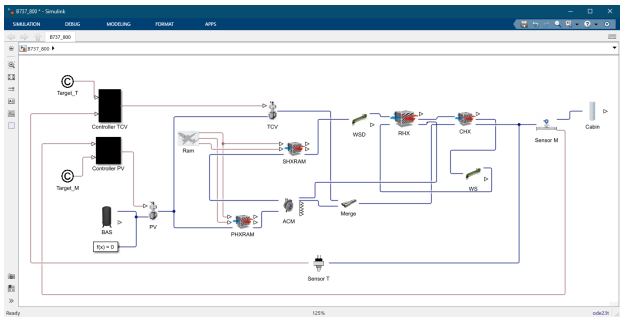

Figure 6. 737NG ECS Model

decoupled models, they may be executed against in-service data at a scale consistent with fleet-level health management activities.

4. Case Study: 737NG Environmental Control System

The methods presented in this work were employed to facilitate increased understanding of failure data of the 737NG Air Cycle Machine with the assistance of an Environmental Control System (ECS) Pack model of the 737NG (Jennions, Ali, Miguez, & Escobar, 2020) in conjunction with in-service data. This model, pictured in Figure 6, uses the Matlab Simscape libraries to simulate the thermodynamic behavior of the air conditioning system on the 737. Executing this model at scale first requires identifyign the input/output interface and exporting the Matlab model to a single executable.

This model requires several steady-state inputs to execute. Some inputs are recorded by the in-service aircraft, while others must be inferred. Based on these boundary conditions, the model simulates the thermodynamics of the ECS heat exchangers and Air Cycle Machine, including inlet and outlet temperatures and pressures. These model parameters are summarized in Table 3. Inputs include Ram air inlet conditions and bleed air properties in addition to fouling factors for the heat exchangers and air cycle machine. Based on these inputs, the temperatures at each component and the pack outlet are calculated. Note that the fouling factors are not recorded in flight parametric data and must be inferred based on the other available information. This inference may be set up similarly to the example given in Figure 4, where the genetic algorithm is used for optimization in this example.

With the basis model and interface parameters defined, the Matlab/Simulink C-Coder application facilitates the export and compilation of the Digitial System Model as described in Section 2.1. This process decouples the model from the Matlab environment and ensures compatibility with the interface requirements. This, in turn, facilitates execution on a large volume of fleet-level aircraft data.

Once the model is exported, it may be invoked in the command line with the input parameters as arguments (in a Unix environment):

```
$ ./B737_Model --Ram_Alt 9133         \
    --BAS_Po 137895 --Cabin_Po 84116 \
    --Target_T_Constantvalue 295.37  \
    --Ram_Ma 0.71    --CHX_n 0.95     \
    --RHX_n 0.95     --SHXRAM_n 0.95  \
    --PHXRAM_n 0.95 --ACM_nm 0.8      \
    --BAS_To 452
```

The model then returns the simulated outputs via the command line like so:

```
Cabin_A_T: 295.370000
PHXRAM_Hi_T: 452.000000
PHXRAM_Ho_T: 282.456881
SHXRAM_Hi_T: 319.940136
SHXRAM_Ho_T: 262.263248
ACM_Ti_T: 277.731433
ACM_To_T: 230.877365
```

By aggregating a large volume of input conditions from various flights, this compiled model can be integrated into the scalable execution framework introduced in Section 3. Using the Baseline Digital Twin Realization in Figure 3 as a reference, the unrecorded model inputs (CHX_n, RHX_n, SHXRAM_n, PHXRAM_n, and ACM_nm) are calibrated based on the recorded model outputs (Cabin_A_T, PHXRAM_Hi_T, PHXRAM_Ho_T, SHXRAM_Hi_T, SHXRAM_Ho_T, ACM_Ti_T, and ACM_To_T). The recorded model inputs (Ram_Alt, BAS_Po, Cabin_Po, Target_T_Constantvalue, and Ram_Ma) are used as the baseline inputs for each episode. This process generates additional, model-based data beyond the recorded values. These data are then used to enrich the dataset to predict component health.

With the digital twin strategy established, a dataset of 44 flight sequences from 41 different aircraft is used to demonstrate the use of this methodology for health management-related activities. These flight sequences are correlated with maintenance data to determine when the Air Cycle Machine was replaced on a given Pack for the aircraft. In total, 2350 flights were analyzed. Of which, 1400 occurred near component removal, while 950 were considered “healthy.”

Executing this model at scale reveals the benefits of the cloud computing framework. The execution time of the 737 ECS Digital System Model in the local Simulink environment is compared against the parallelized cloud computing environment in Figure 7. As the number of simulations increases in the local environment, the simulation time increases substantially. By comparison, parallelization substantially reduces the time required to execute a large number of simulations in the cloud computing envrionment, increasing the throughput

Name	Input/Output	Description	Recorded?
Ram_Alt	Input	Aircraft Altitude (m)	Yes
BAS_To	Input	Bleed air outlet temperature (Kelvin)	Yes
BAS_Po	Input	Bleed air outlet pressure (Pa)	Yes
Ram_Tat	Input	Ram inlet temperature (Kelvin)	Yes
ACM_Ti_T	Output	Air Cycle Machine turbine inlet temperature (Kelvin)	Yes
ACM_To_T	Output	Air Cycle Machine turbine outlet temperature (Kelvin)	Yes
PHXRAM_Hi_T	Input	Primary Heat Exchanger inlet temperature (Kelvin)	Yes
PHXRAM_Ho_T	Output	Primary Heat Exchanger outlet temperature (Kelvin)	Yes
SHXRAM_Hi_T	Output	Secondary Heat Exchanger inlet temperature (Kelvin)	Yes
SHXRAM_Ho_T	Output	Secondary Heat Exchanger outlet temperature (Kelvin)	Yes
Cabin_Po	Output	Pack outlet pressure (Pa)	No
Cabin_A_T	Output	Pack outlet temperature (Kelvin)	Yes
CHX_n	Input	Condenser fouling factor	No
RHX_n	Input	Reheater fouling factor	No
SHXRAM_n	Input	Secondary Heat Exchanger fouling factor	No
PHXRAM_n	Input	Primary Heat Exchanger fouling factor	No
ACM_nm	Input	Air Cycle Machine mechanical fouling factor	No

Table 3. 737 NG Parameters

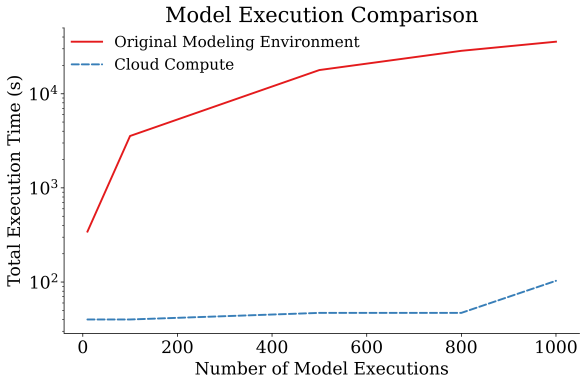

Figure 7. Model Execution Time Comparison

	Positive Sequences			Negative Sequences	
Method	Balanced Accuracy	Recall	Precision	Balanced Accuracy	Average Accuracy
Recorded Inputs	50.6%	56.9%	28.6%	94.7%	72.7%
DTw Data with Inputs	54.2%	53.0%	26.4%	97.9%	76.0%
DTw - Recorded	3.6%	-3.9%	-2.2%	3.2%	3.3%

Figure 8. 737 ACM Removal Model Performance

and facilitating execution of this small-scale fleet-level experiment.

Using this representative dataset, a random forest classification model was trained to predict whether a 10-flight window, sampled from either an "unhealthy" (positive) sequence or "healthy" (negative) sequence would end with component removal. This process was executed twice: once with only the recorded parameters and once with the recorded parameters in addition to values generated from the model simulation. Using cross validation to determine hyperparameters, the testing results are shown in Figure 8. As shown, the additional information generated by the DTw improves the balanced accuracy of the removal model. While both models performance is insufficient for deployment, this demonstrates the intended use-case for this methodology.

5. Conclusion

This work has presented a comprehensive strategy for the development and utilization of Digital System Models and Digital Twins for aircraft health management. Using this stragety ensures that the models may be used at-scale for fleet-level prognostics development. A Simulink model of the 737NG Environmental Control System Pack was used to show how this approach may be used to assist with these prognostics activities.

References

Hatakeyama, J., Seal, D., Farr, D., & Haase, S. (2018). Systems engineering v in a model-based engineering environment: Is it still relevant? In *2018 aiaa space and astronautics forum and exposition* (p. 5326).

Jennions, I., Ali, F., Miguez, M. E., & Escobar, I. C. (2020). Simulation of an aircraft environmental control system. *Applied Thermal Engineering*, *172*, 114925.

Mihai, S., Yaqoob, M., Hung, D. V., Davis, W., Towakel, P., Raza, M., ... others (2022). Digital twins: a survey on enabling technologies, challenges, trends and future prospects. *IEEE Communications Surveys & Tutorials*.