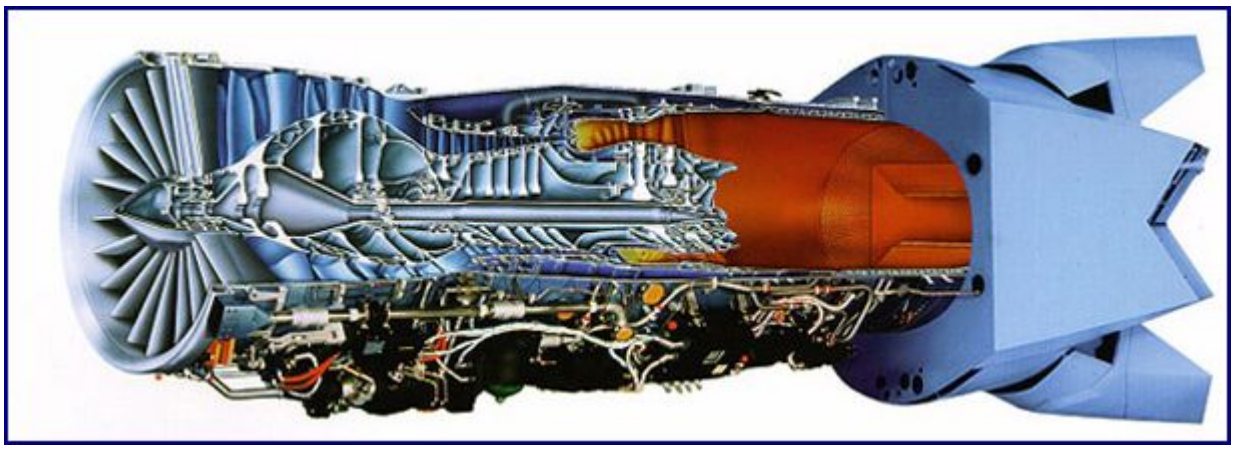
MODELING & SIMULATION OF A TURBOJET ENGINE FOR FUTURE COMMERCIAL SUPERSONIC TRANSPORT UTILIZING THE DEVS-SUITE EXPERIMENTAL FRAMEWORK



Lucio Ortiz
area of study Modeling and Simulation

Robert Blazewicz
area of study Embedded Systems

An Applied Project Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Engineering

(Submitted for Approval) by the
Graduate Supervisory Committee:
Hessam Sarjoughian, Chair

ARIZONA STATE UNIVERSITY

2016-12-02

School of Computing, Informatics, and Decision Systems Engineering
Arizona State University
Tempe, AZ, USA, 85281

ABSTRACT

Aeronautics have advanced at an increasing rate over the last century. One of the paramount achievements was reaching flight speeds exceeding the speed of sound, supersonic flight! With the turn of the century, and the advances in aircraft design, as well as aeronautic research and development, recent interest has risen once again to investigate a feasible solution to supply commercial aviation with aircraft capable of supersonic flight [1]. Recent advances in research and development of low boom, supersonic flight over land are now underway, and this research has proven to show promising results [2]. The benefits would allow flight from one side of the country to the other in a fraction of the time versus conventional, subsonic aircraft.

With this in mind, one area that will need to be addressed is the design of aircraft engines, capable of thrust powerful enough to allow supersonic flight for large commercial aircraft; however, with the concern of fuel costs and efficiencies in mind. Supersonic flight has been catered to military aircraft, and the needs and capabilities of military use. These involve very extreme flight regimes, including very high speed flight with much maneuverability. Supersonic flight in most military aircraft use both the main burner and afterburner of modern turbojet engines. Although fuel efficiency is a concern for military requirements, it comes almost secondary to the capabilities of the aircraft in combat missions. The strategies developed must be feasible for commercial supersonic aircraft.

This project intends to expand on the research towards this goal. With the use of the discrete-event / discrete-time simulator, DEVS-Suite, provided by the Arizona State University – Arizona Center for Integrative Modeling and Simulation, a turbojet engine is to be modeled and simulated for high-speed flight in order to offer insight into cost savings for future, feasible, commercial supersonic aircraft. The ultimate intent is to provide information on how to best achieve sustained flight at supersonic speeds while keeping fuel costs low.

A second portion of this project is to develop an extension to the DEVS-Suite for performing Design Experiments on models like the turbojet engine, so that important features of a test model may be evaluated and the generated output available for analysis. Utilizing the new Experimental Framework, the goal is to offer a research tool which, allows this software to be adapted to many other complex system-of-system (SoS) models. We provide the start of a robust tool that allows users to analyze complex control system models for analysis.

A detailed description follows as well as the approach to be taken to solve the necessary problems within this modeling and simulation effort.

Keywords: Modeling & Simulation, DEVS, Discrete Event System Specification, Turbojet Engine, Supersonic Flight, commercial supersonic flight

TABLE OF CONTENT

Modeling & Simulation Development Platform

DEVS-Suite Simulator, 2015 — 3.0.0
The Arizona Center for Integrative Modeling and Simulation
http://acims.asu.edu/software/devs-suite/

INTRODUCTION

Looking at the cutaway figure on the cover page of this document of a turbojet engine, one can see the complexity of all the separate systems involved, working together, to ultimately achieve the end result of thrust. In order to begin modeling and simulating this system of systems, a higher level of abstraction will initially need to be taken, in order to reduce some of the complexity of the model, yet still provide a high enough resolution of which the data gathered will be useful. The components in the abstraction layers can then be separated even further as expertise and research allows, to provide in-depth details which will enhance the accuracy of the model. We will follow George Klir's system framework [3] as noted in Table 1 below, and develop our model specifications. The idea here is to break the system of systems down into smaller components, then couple these together to represent the complete turbojet system. A schematic of the initial component breakdown is shown below in Figure 1.

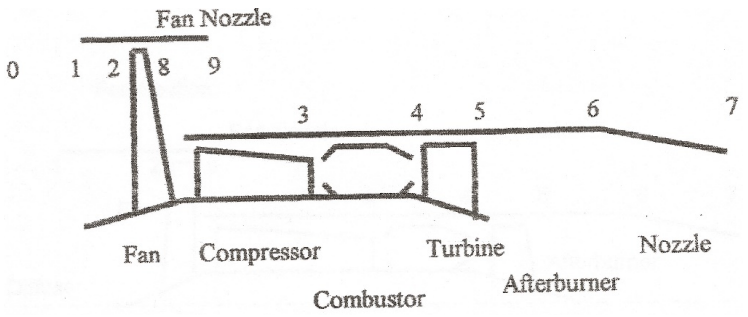


Figure 1: Model of Turbojet Components & Stages

The main components shown above are complex, and each will begin at some level of abstraction as well, yet show enough behavioral characteristics so as to provide a sufficient model and analysis of the simulation.

In order to achieve our goal of developing a simulation which provides information on the thrust output from a turbojet, and analyze the fuel consumption, we will need measurable characteristics. We will thus follow the thermodynamic flow characteristics of (air) mass and fuel flow through the various stages of the turbojet aircraft engine, and analyze how the results of thrust and fuel flow are affected by varying the input parameters. A sample set of stages is represented in Figure 1 above with inputs/outputs traversing between components, (shown by numbered stages of the system). As was mentioned earlier, the intent is to provide aircraft engines for supersonic commercial transport over land, and thus the input parameters will be the flight speed (Mach number), which will give us the information of the air mass flow, as well as initial

characteristics of the turbojet engine components, (initial states). The measured, output results will be the characteristics of the flow and the effects of the various flight speeds and be measured by (specific) thrust, and Thrust Specific Fuel Consumption (TSFC), in addition to the turbojet efficiencies.

Motivation for DEVS-Suite Experimental Framework

Reflecting once again on our goals for this project, we restate that we want to be able to model a complex system of systems, and simulate a complex system of systems. At this point in time, we have no idea what the model is to look like. This gives no knowledge to the simulation portion of our framework, and thus no direct couplings can be formed. In fact, this is exactly as we want to proceed. Decoupling of the model from the experimentation (simulation) of the framework is key to providing a research tool which can accommodate any complex system of systems. If there was a hard-coded model architecture in which the simulator was limited to, there would impose much limitation on the capability and flexibility of the research framework. We aim to offer the most flexible, maintainable, and capable research framework for any type of complex network of system interactions.

We have researched some design tools, more specific to the goal of simulating our turbojet engine and have decided that there are either limitations, restrictions on the design/development/alteration of the models, or tools that may exist but are not readily available as research tools, (i.e. In-house, private simulation tools). The American Institute of Aeronautics and Astronautics does provide a somewhat capable tool for turbojet engine research, however, this is not a fully automated tool for simulations in which we desire, (i.e. does not allow the flexibility to design one's own models, add/manipulate models for optimization, or otherwise customize the modelling process). The tool is very capable to provide verification data to eventually compare the simulation data from the proposed framework, yet to be designed, and allow a measurement of accuracy.

Problem Approach

It is with the previously mentioned information that we choose to look at the DEVS-Suite Experimental framework to provide the platform for our research tool. The modeling formalisms for a complex network of system components will not necessarily be known at the onset of a project initiation. We need a modeling and simulation platform which can at least accommodate each basic modeling formalism and simulate them appropriately, (i.e. discrete time (DTSS), continuous (DESS), and discrete event systems (DEVS)). The DEVS-Suite Experimental framework is inherently designed to allow "Formalism Transformation" [4]. Formalism transformation allows the mapping from one formalism to another, either to add efficiency, or the details lacking in one formalism to appropriately describe the model and its environment correctly. An interpretation of these different types of mappings can be seen below in Figure 2. [5].

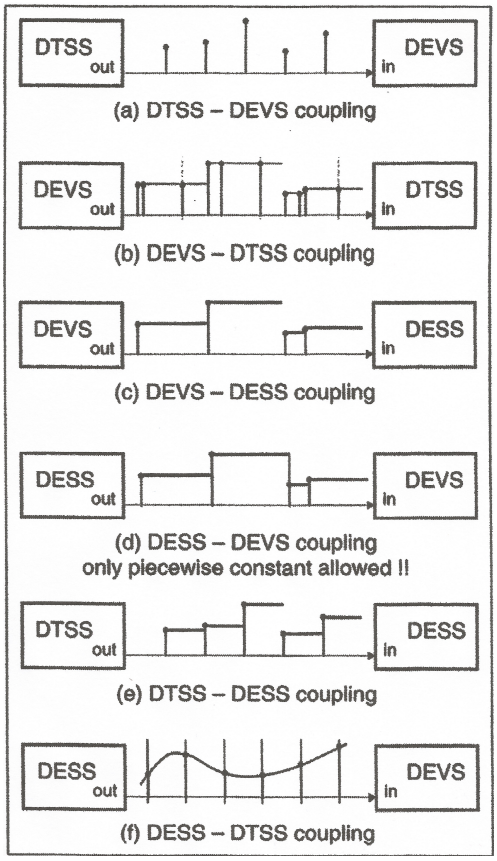


Figure 2: Multiformalism System Coupling Interpretations

As we can see from the mappings, if our framework is able to accommodate all three basic formalisms, and map from one to another, then we can design our system model, and its component models with the flexibility of using discrete time, differential equations (continuous), or a discrete event system specification. The discrete event system specification, as we will explain soon, provides unique advantages of common abstraction methods which offer the benefits of a broader range of experimental exploration and, in most cases, increased simulation efficiencies. Hence, the scope of the project will include a DEVS model and simulation approach, utilizing the DEVS-Suite Experimental framework, modeling and simulation application. We can now begin to formulate our model specification, (as noted in Table 1), following George Klir's system framework [3].

The fundamental types of problems for systems in modeling and simulation will be considered for this project, according to Zeigler, Praehofer and Kim in "Theory of Modeling and Simulation, 2nd Ed." [5]. Three fundamental types attempt to classify a system. In the case of the turbojet engine for supersonic commercial transport, the system falls partially into all three categories and thus, each domain will be explored.

Table 1: Levels of System Knowledge

Level	Name	What we know at this Level	Turbojet Simulation
0	Source	What variables to measure and how to observe them	• Thrust Specific Fuel Consumption (TSFC), Specific Thrust output, and Turbojet Efficiencies. • Observed through various stages of flight (input Mach #'s), including sustained supersonic speeds.
1	Data	Data collected from a Source System	• Flight data and engine output data from research • Comparison of turbojet output measurements with AIAA Simulation Software.
2	Generative	Means to generate data in a Data System	• DEVS Model to recreate engine operation and current and adjusted input flow characteristics.
3	Structure	Components (at lower levels) coupled together to form a generative system	Turbojet Components • Inlet (including Fan) • Compressor • Combustor • Turbine • (Afterburner) • Nozzle

From a systems analysis perspective, turbojet propulsion for supersonic flight is not entirely new, however, as mentioned earlier, is most primarily used for military aircraft. We will take from what the military has discovered and attempt to recreate a similar, baseline system to which modifications will later be applied. We want to understand the behavioral characteristics clearly in order to make strategic decisions about any future modifications.

From a systems inheritance perspective, as the system exists from the systems analysis, we will now vary factors in order to observe changes in behavior. At this point, it may be necessary to take a "design of experiments" approach and choose factor interactions systematically in order to eliminate an exhaustive, and unfeasible set of all interaction possibilities.

Finally, from the insight gained in the systems analysis and the systems inheritance, the systems design view will look to provide insight as to enhancements, or flight regimes that achieve the goal of reduced fuel usage for sustained supersonic flights. Table 2 details the fundamental systems problem approach to be taken.

Table 2: Fundamental Systems Problems

Systems Problems	Does source of the data exist? What are we trying to learn about it?	Which level transition is involved?	Turbojet Simulation
Systems Analysis	The system being analyzed may exist or may be planned. In either case we are trying to understand its behavioral characteristics.	Moving from higher to lower levels, e.g., using generative information to generate the data in a data system.	Turbojet engines for supersonic flight exist, utilized in military applications. The first step is to attempt a recreation of these engine characteristics and operations under supersonic flight conditions to observe an initial capability baseline.
Systems Inference	The system exists. We are trying to infer how it works from observations of its behavior.	Moving from lower to higher levels, e.g., having data, finding a means to generate it.	We will next try to understand what affects the behaviors of fuel consumption and thrust output during various supersonic flight regimes by adjusting factor (variables) in our model.
Systems Design	The system being designed does not yet exist in the form that is being contemplated. We are trying to come up with a good design for it.	Moving from lower to higher levels, e.g., having a means to generate observed data, synthesizing it with components taken off the shelf.	Finally, our model will provide insight as to the turbojet engine enhancements that may provide for efficient, low-fuel use, commercial supersonic transport aircraft.

Before we begin our Model Design, we would like to emphasize one more advantage of using the DEVS-Suite Experimental Framework for our problem approach. One important concept of Model & Simulation (M&S) Theory is to decouple, or separate, the model design from the experimental framework design as much as possible. To reiterate, this allows the models to be designed completely independent from the simulation framework, such that any model, designed with any formalism, should be able to be simulated by the framework. This offers a re-usable experimental framework, and the ability to "swap" in and out component models, and system models alike, as research sees fit. We can think of this concept of M&S Theory as a "Unified Model and Experiment Design Structure", shown in Figure 3.

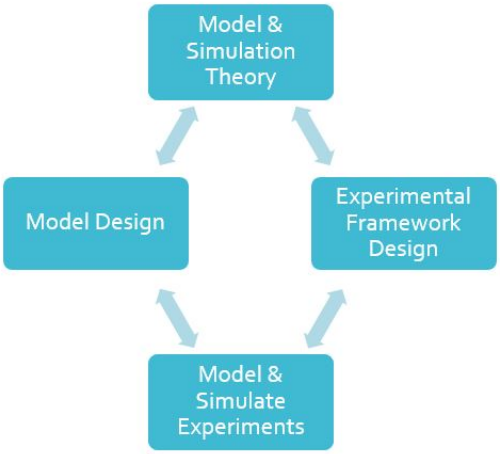


Figure 3: Unified Model — Model Design

We can see from the above diagram where the individual design of the models, separate from the framework, both derive from M&S Theory, however, join together to ultimately define the M&S Experimental Framework, capable of simulating the Models. We can now move to design our models, both for the system components and for the Turbojet Engine System of Systems.

MODEL DESIGN

At this point, we are ready to begin our systems specification (see Table 3), and in particular, the general turbojet engine system component specification. Recalling the "System Specification Hierarchy" [7], we begin with the Observation frame, and will move towards the detailed level of Coupled Components.

Table 3: System Specification Hierarchy

Level	Specification Name	Corresponds to Klir's	What we know at this level
0	Observation frame	Source System	How to stimulate the system with inputs; what variables to measure and how to observe them over time base.
1	I/O Behavior	Data System	Time-indexed data collected from a source system; consists of input/output pairs.
2	I/O Function		Knowledge of initial state; given an initial state, every input stimulus produces a unique output.
3	State Transition	Generative System	How states are affected by inputs; given a state and an input what is the state after the input stimulus is over; what output event is generated by a state.
4	Coupled Component	Structure System	Components and how they are coupled together. The components can be specified at lower levels or can even be structure systems themselves-leading to hierarchical structure.

Model System Specification

Following the DEVS formalism, we can choose a discrete time base and several input variables to stimulate the system. In the case of the turbojet engine as a complete system, the input variables are directly related to the environment to which the turbojet engine is exposed to, (i.e. flight regime). In the case of the individual components, the input variables are also related to the flight regime they are expose to, but also related to how a previous or aft system component is operating and the input information that is received from them. We can generalize our inputs and outputs, including the time base for either the system as a whole, or for an individual component as an Observation Frame. Focusing on any individual system component within the turbojet engine system, we can represent our Input/Output (I/O) Observation Frame as follows:

$IO = \langle T, X, Y \rangle$

$T = \{ t \mid t \in \mathbb{N}_0^+ \cup \infty \}$

$X = \wp \{ v_i \mid v \in \text{Input Component Variables} \}$, $\wp$: Power set

$Y = \wp \{ v_i \mid v \in \text{Output Component Variables}, , \varphi \}$, $\wp$: Power set

Here, for any system component, we still have a black box abstraction. We can define our input variables, and which variables we would like to measure as the output variable sets. With the time base, we can record a set of output variable behaviors over time in response to the inputs. This moves us into a system specification at the behavioral level, or "Level 1". At this behavioral state, we can increase knowledge with the concept of input and output segments. The input segments (X,T) and the response to the input segments (Y,T) share the same observational frame, such that the observed input-output pair are recorded over the same interval and the collection of several input-output pairs over several experiments is considered a relation, or I/O Relation. The set of all relations defines all possible input/output behaviors of the system (component). At this level of system knowledge, I/O Relation, we still have a black box representation of our system components. The system response is unpredictable and ambiguous since it depends on the inner workings of the black box, of which we need more knowledge to define. We thus move to introduce the I/O Function Observation.

The I/O Function Observation (IOFO) partitions the relations by means of a set of functions $F=\{f_1, f_2, ...f_i...\}$ to which each input segment produces a unique output segment. These set of functions thus represent the "initial state" of any system component. Given the initial state defined by the set of functions, F, we can insert an input segment to the system component and observe a unique output response, an output segment. Though we have knowledge of the initial state of the system, we are not yet able to visualize any of the

immediate and final states of the system, however, the IOFO is an important step in the model design. From here, we can start to design our model more specific to our needs. We have access to knowledge to design the behavior of the system component to accurately represent the component in the system. Here we can specify our IOFO structure as follows:

IOFO = 〈 *T*, *X*, *Y*, *Ω*, *F* 〉

Where T, X, Y, are as before in the I/O frame, *Ω is the set of allowable input segments,*

$T = \{ t \mid t \in \mathbb{N}_0^+ \cup \infty \}$

$X = \wp \{ v_i \mid v \in \text{Input Component Variables} \}$, $\wp$: Power set

$Y = \wp \{ v_i \mid v \in$ Output Component Variables, , φ $\}$, $\wp$: Power set

$\Omega \subseteq (X,T)$ – the set of allowable input segments

and,

$F = \{f_1, f_2, \ldots f_i \ldots\}$, where

$f \in F \Longrightarrow f \subseteq \Omega$ x (Y,T) is a function

and if

$$\rho = f(\omega), then\ dom(\rho) = dom(\omega), where\ \omega = input\ segment, \rho = output\ segment$$

Following the IOFO structure, an example turbojet engine system component IOFO Specification is as follows, (for the Diffuser Inlet of the Turbojet Engine):

Stage: 1 to 2 Diffuser IOFO = 〈 *T*, *X*, *Y*, *Ω*, *F* 〉 :

$T = \{ t \mid t \in \mathbb{N}_0^+ \cup \infty \}$

Assumptions:

Flow is decelerated with no loss in stagnation pressure and no change in stagnation temperature.

Known (Inputs):

$X = \wp \{ v_i \mid v \in \text{Input Component Variables} \}$, $\wp$: Power set

Input Component Variables:

- Flight Speed (V_0)
- Mach # (M_0)
- Flight (pressure) altitude

- Ambient (static) air parameters (T_0, P_0, ρ_0), based on pressure altitude
- Ratio of specific heats, (γ)

$\Omega \subseteq (X,T)$ – the set of allowable input segments

F={f_1 , f_2,…f_i…}, where f = {Component Functions}

Component Functions:

$$a_0 = \sqrt{\gamma R T_0}$$ (pressure altitude) speed of sound 1.1

$$T_2^\circ = T_1^\circ = T_0^\circ = T_0(1 + \frac{\gamma - 1}{2} M_0^2)$$ Total (stagnation) temperature at Diffuser Inlet 1.2

$$P_2^\circ = P_0(1 + \frac{\gamma - 1}{2} M_0^2)^{\frac{\gamma}{\gamma - 1}}$$ Total (stagnation) pressure at Diffuser Inlet 1.3

$$\tau_r = \frac{T_0^\circ}{T_0} = \frac{T_2^\circ}{T_0}$$ adiabatic freestream recovery enthalpy ratio 1.4

$$\pi_r = \frac{P_0^\circ}{P_0} = \frac{P_2^\circ}{P_0}$$ Isentropic freestream recovery pressure ratio 1.5

Outputs:

$Y = \wp$ { v_i | v ∈ Output Component Variables, , φ }, $\wp$: Power set

Output Component Variables:

$a_0, \tau_r, \pi_r, T_2^\circ, P_2^\circ$

Similarly, we specify the IOFO for each of the system components in the Turbojet Engine System. (A complete set of the IOFO Specifications for each Turbojet Engine Component was created for our project.)

With the behaviors (functions) specified for our system components, we are ready to combine the components together to ultimately achieve a specification for a complete “System of Systems”. This leads us into our experimental frame definitions and further into how these models will ultimately be simulated in our framework.

The Discrete Event System formalism takes advantage of event-based state changes. The changes only of interest to the system environment to which the component is submersed in, are the only state changes that a discrete event system models. This increases computational efficiency, and simplifies the model and simulation, while still capturing the important, and relevant events. We continue our system specification then, following the DEVS approach and further add knowledge to the interior of the system components by

the addition of state sets and state transitions, (functions). Expanding the IOFO, the state set adds knowledge of the past, such that the future can be calculated, given the current state and some future input, a property called composition. The system specification formalism then becomes:

I/O System: $S=(T, X, Y, \Omega, Q, \Delta, \Lambda)$*

$where\ T, X, Y, and\ \Omega$ are the same as the IOFO and

$Q\ is\ the\ state\ set,$

$\Delta: Q \times \Omega \rightarrow Q$ is the $global$ state transition function,

$\Lambda: Q \times X \rightarrow Y$ (or $\Lambda: Q \rightarrow Y$) is the $output$ function.

*with the constraints of closure property and composition property.

In this general form, and from the theory of modeling and simulation levels of knowledge, by applying a state set and state transition functions to any known IOFO structure, we can now obtain a formal specification at a system level. Thus, from the previous section, if we know the state sets for each of our turbojet engine components defined in IOFO, we can create *system components,* more formally, and directly. This goes without saying that, for any *system component*, we can determine its IOFO.

Before further graduating this abstract I/O System, we can take the time to mention, that this specification can now be formalized into many system types, based on the behavior we desire to model. Time-Invariant systems, Input-Free and Memoryless systems, and generator-system types can all be created from this abstract specification [4].

The abstract system can be applied to define a single turbojet system component, any component. The contribution of all the components, working together to ultimately produce thrust, form a network of system components. We can group our system components together, and pass information between them which will define the overall behavior of the entire turbojet engine. In fact, for any system of systems, this concept works quite well in modeling and simulating a complex system. We can define a set of *component references* D = {Component References}, such that for all $d \in D$, the system components can be formed by

$M_d = \langle T, X_d, Y_d, \Omega, \Delta, \Lambda \rangle$

where M_d is an I/O System.

We can now group these components into either a *multicomponent system* or *a coupled system* specification. In a multicomponent system, each component contributes to the overall behavior of the network system through its state transitions. Multicomponent systems have a limitation, however, of only being able to change their own state when defined by DTSS or DESS formalisms. In order to maintain a flexible set of

modelling, we opt to use a *coupled system* specification for our network system. This allows each component to be able to influence, (change) the state of other components, through its input and output interfaces. This offers the advantage of modularity, and forms a sense of influencer/"influencee" relationships among the system components. Further, through M&S Theory, if a system in a network of systems, is itself specified in a formalism, and if the combined network of systems is also, specified in the same formalism, then the network and each component system are considered *closed under coupling* and form a valid system of systems. We can immediately envision a component included in larger groups of components to form a system, and even this system, grouped together with other systems to form yet, even more complex system of system models. Our network takes the form:

N = 〈 T, X_N, Y_N, D, {M_d | $d \in D$}, EIC, EOC, IC 〉 ([4] Zeigler, Praehofer, & Kim, 2000)

$X_N = \{IPorts_N, X_{N1} \text{x } X_{N2} \text{x} X_{N3} \dots\}$ is the set of multivariable set of inputs of the network with variables $\{IP_N \in IPorts_N\}$

$Y_N = \{OPorts_N, Y_{N1} \text{x } Y_{N2} \text{x} Y_{N3} \dots\}$ is the set of multivariable set of outputs of the network with variables $\{OP_N \in OPorts_N\}$

D is the set of "turbojet engine" components, where d ∈ D is a system, as mentioned previously:

$M_d = (T, X_d, Y_d, \Omega, Q, \Delta, \Lambda)$ = Turbojet Engine Component "Systems"

With $X_d = \{IPorts_d, X_{d1} \text{x } X_{d2} \text{x} X_{d3} \dots\}$, and $Y_d = \{OPorts_d, Y_{d1} \text{x } Y_{d2} \text{x} \dots\}$ the respective input and output ports of each component d ∈ D.

$EIC \subseteq \{((N, ip_N), (d, ip_d)) | ip_N \in IPorts_N, d \in D, ip_d \in IPorts_d\}$ is the external input coupling

$EOC \subseteq \{((d, op_d), (N, op_N)) | op_d \in OPorts_d, d \in D, op_N \in OPorts_N\}$ is the external output coupling

$IC \subseteq \{((a, op_a), (b, ip_b)) | ip_b \in IPorts_b,\ a, b \in D, op_a \in OPorts_a\}$ is the internal coupling

With the constraints that the range of the to-port must be a subset of the range of the from-port:

$\forall((N, ip_N), (d, ip_d)) \in EIC: range_{ipN}(X_N) \subseteq range_{ipd}(X_d)$

$\forall((d, op_a), (N, op_N)) \in EOC: range_{opd}(Y_d) \subseteq range_{opN}(Y_N)$

$\forall((a, op_a), (b, ip_b)) \in IC: range_{opa}(Y_a) \subseteq range_{ipb}(X_b)$

Continuing further from ([4] Zeigler, Praehofer, & Kim, 2000):

The set of influencers $\{I_d\}$ and interface map $\{Z_d\}$ can be defined,

$$I_d \subseteq D\ \{N\}\ is\ the\ set\ of\ influencers\ of\ d, and$$
$$Z_d\colon x_{i\in I_d} YX_i \to XY_d\ is\ the\ interface\ map\ for\ d$$

where the set of influencers $\{I_d\}$ of all components $d \in D$ and Network N is derived by gathering all components from which a coupling to d or N exists

$$\forall d \in D \cup \{N\}\colon I_d = \{i | \big((i, fromport), (d, toport)\big) \in EIC \cup EOC \cup IC\}$$

and the map $\{Z_d\}$ is a function which always takes the value of the $fromport$ of that coupling which is equal to the nonevent $\emptyset$, if such a port exists, and

$$\forall d \in D \cup \{N\}\colon Z_d\colon X_{i\in I_d} YX_i \to XY_{d,}\ with\ Z_d\big(\dots, yx_{i,fromport}, \dots\big) = xy_{(d,toport)}, where\ xy_{(d,toport)}$$
$$= \left\{ \left\{ \bigoplus_{(i,fromport) \mid ((i,fromport),(d,toport)) \in EIC \cup EOC \cup IC} \right\} yx_{(i,fromport)} \right\}$$

In this system specification structure, we can note the influencer/"influencee" relationships, the individual state sets and state transitions which each component contains, and the couplings and modularity which makes this a powerful strategy to modeling a complex system of systems. Further, the formalisms, (i.e. DESS, DTSS, and DEVS) are preserved to be used in any component, and they can still be linked, as was previously mentioned and shown in Figure 2: Multiformalism System Coupling Interpretations. One further advantage of this set of specifications, as we will see, is that the architecture can also employ the advantages of abstraction and inheritance as in an object orientation architecture and system components can be built from a "base class" at a higher level of abstraction, simply by adding the details necessary to further define the specific system component.

With the models formalized, we are ready to move into the "Experimental Framework Design", however, before we do, it would be beneficial to consider what our experimental goals will be and return to our roadmap of Modeling & Simulation, as shown once again in our "Unified Model and Experiment Design Structure" (Figure 4).

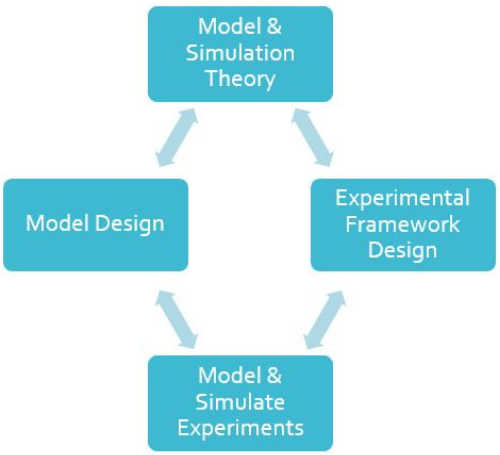


Figure 4: Unified Model — Model & Simulation Theory

MODEL AND SIMULATION THEORY

Theory of Experimental Framework Design

The "Experimental Framework Design" derives from Model and Simulation theory, and so we revert a bit to examine our experimental needs. We can simply analyze our needs through our goals.

- Turbojet performance characteristics and behavior are of primary interest in our specific system of systems.
- A future platform for research of any complex system is desired.

Utilizing a traditional model and simulation approach, whereas (from the model specifications), time and event trajectories are of specific importance to the simulation. This "time view" approach offers information into state transitions and the timing of these events. An experimental frame in this sense, is an observer of the segments. Traditionally, an experimental frame in this context has three types of components, a generator, acceptor, and transducer. The generator, as its name suggests, generates input segments to the system, (system component or system of systems). The acceptor is a "monitor" of the experiment, and assures that the desired conditions are met within the experimental frame. The transducer in this implementation is the "observer/analyzer" and analyzes the system output segments. In our experiment goals, however, we are interested in more than just when a state change happens, or if it happens, and what the output trajectories look like. Further, we do not necessarily desire our experimental system behavior to be governed by the "time view" Our experimentation efforts call for the gathering of the information outputs of the behavior of the system, and how to optimize it. We can thus have different experimental frames for different purposes. In order to accommodate both our complex system experimental goals, and leaving flexibility for a traditional "time-view" simulation, we can look to base our experimental frames from a traditional standpoint, and later enhance our framework as necessary to allow for the capabilities which we

desire. This will form a more "holistic" approach to our model and simulation efforts and, in particular, a holistic "view" of our "Experimental Framework Design" specifications.

These concepts are now discussed in the following sections, as we begin with the more formal, and basic theory of experimental frames which can be used in a "time-view" implementation, however, have limitations to provide all the requirements of experimental simulations. These basic experimental frames, however, pose no limitations to being used in "enhanced" framework designs which will indeed provide the requirements we desire.

"Traditional" Experimental Frames

Considering the DEVS System specifications, we can again refer to (Figure 1: Model of Turbojet Components & Stages), and begin to think about and develop the experimental frames which will be modeled and simulated. First, we need to consider each of the components that make up the turbojet engine. Second, we need to consider the environment, and which environmental variables are most important to realistically simulate the model. Interactions between components and the flow of information, i.e. mass flow parameters including air and fuel flow among others, will be of importance in this simulation to gather the necessary data for analysis, so internal coupling relationships need to be defined. Consideration of DEVS specifications within individual components themselves will also need to be addressed. Finally, when the model is formally designed and ready to be simulated in its environment, consideration of external couplings to the environment will need to be created.

Accommodating the requirements mentioned above, our experimental frames for the engine can begin to be developed, as shown in Figure 5.

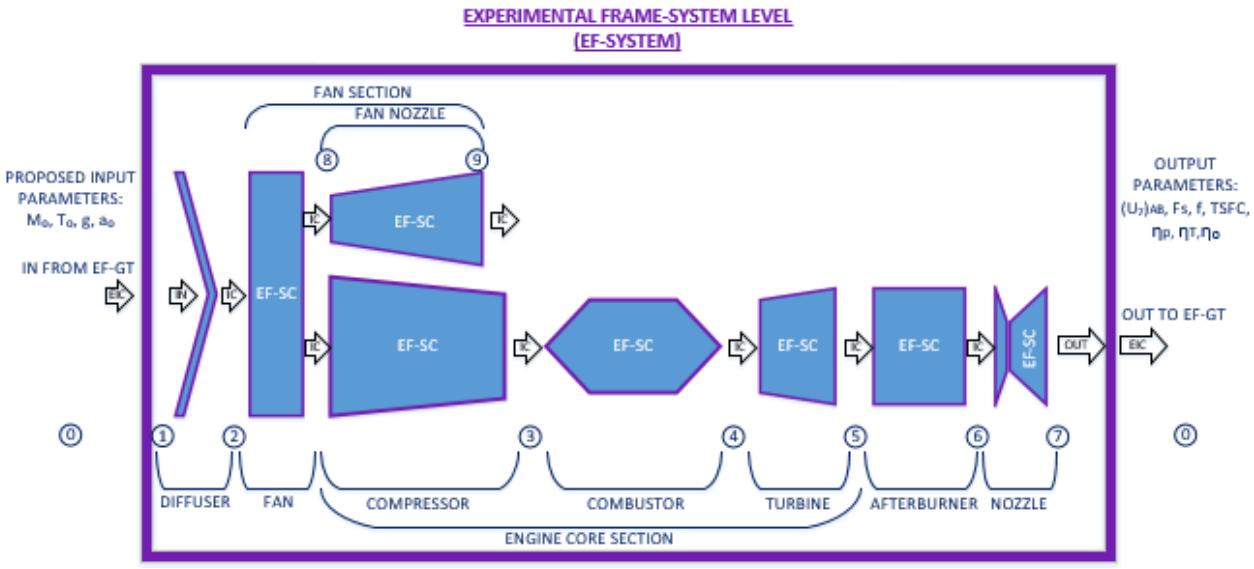


Figure 5: Experimental Frame of Turbojet Engine System

The turbojet engine from a system level perspective forms a hierarchical model experimental frame with each of the engine components as coupled models. At this system level view, we can see the internal couplings between the engine components as they are laid out according to our turbojet engine schematic. In this case, the turbojet engine considered contains a turbofan layout with afterburner. Inlets and outlets between engine components which our flow parameters will traverse, are designated with station numbers. These station numbers provide an organized method for notation in the equations that follow the path of the flow parameters, (a descriptive legend is shown in Figure 6).

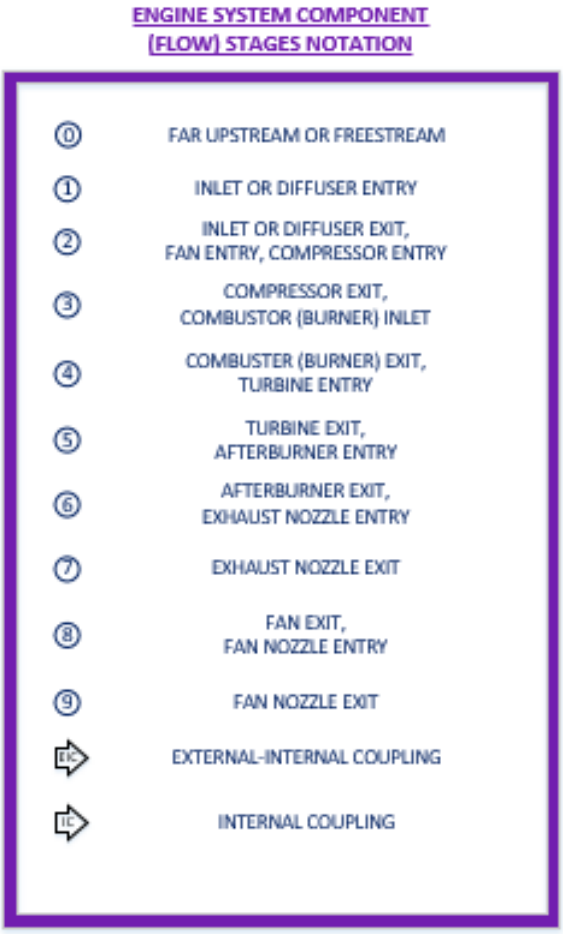


Figure 6: EF-Engine Systems Legend

Each of the engine components will need to be specified as coupled models in themselves and a base model of a simple, system component experimental frame, can be seen in Figure 7.

In a base model design, each component will need a DEVS specification which may include (at least) one processor, and an optional generator and transducer for additional input parameters, specific to the system component being modeled. Figure 7 shows a basic model to which individual components may be built on. In a general system of systems model, and depending on the complexity involved, some components within

a system can include several processors, including pipeline or parallel layouts with optional switches and buffers.

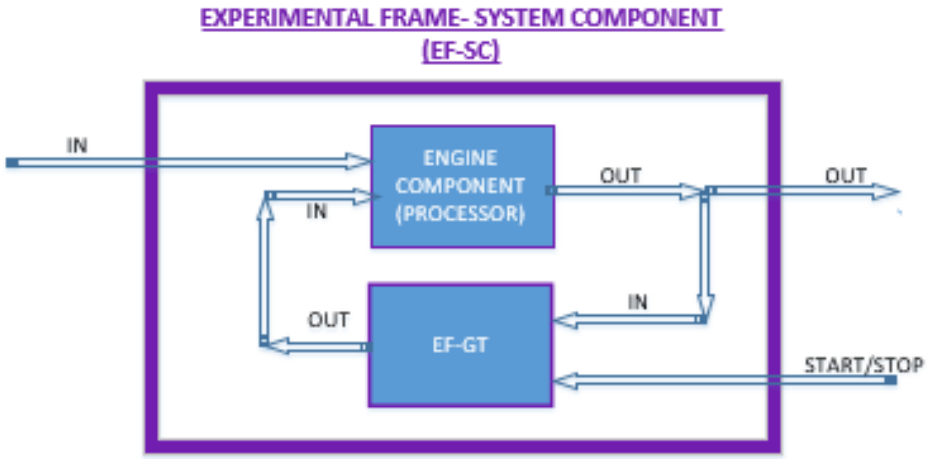


Figure 7: Experimental Frame of Engine System Component

Whether considering a separate, decoupled system component for any system of systems, or an entire system composed of many individual system components, separate generators may be necessary to provide inputs. In order to monitor time or event trajectories, it is encouraged to monitor these inputs via a transducer, and so will include a separately defined experimental frame for a "generator-transducer", (EF-GT). These EF-GT experimental frames can now be customized and coupled within the EF-SC experimental frames, as necessary, or to provide inputs to the system from an external vantage point. The concept of an EF-GT to provide inputs to the system, will also aid in the simulation of the environment to which we will immerse our system into. The EF-GT frames are shown both in Figure 5 providing the input parameters to the system, as well as in Figure 7 as an option for individual system components. A model of the EF-GT is shown in Figure 8.

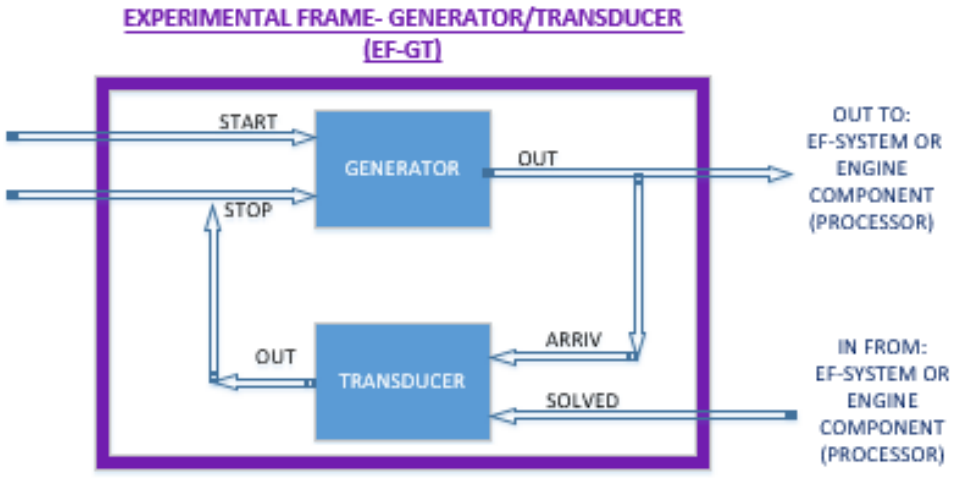


Figure 8: Experimental Frame of Generator-Transducer Components

Taking advantage of the station notations in the nomenclature, the initially proposed input and output parameters are also shown in Figure 5. To analyze the model performance of the turbojet engine, the environmental parameters to which it will operate in are of vital importance to the output efficiencies. During supersonic flight, the environment needs to model the "real" fluid characteristics to which the aircraft will fly in. Thus, the airspeed (freestream Mach number), temperature, and speed of sound (at certain pressure altitude) will be some of the inputs to the system. The outputs gathered will need to measure the performance and efficiency of the turbojet during this supersonic flight. For that we will gather data on the exit velocity of the propulsion system which will allow us to calculate the specific thrust. Since we are looking to provide insight to a more efficient and economical (fuel savings) performance, we will also need to monitor the fuel flow rate, the thrust-specific-fuel-consumption, and the characteristic efficiencies of the propulsion system, the thermal quality, and the overall engine efficiency. A nomenclature list of these proposed inputs and outputs is shown in Figure 9.

ENGINE SYSTEM INPUT/OUTPUT PARAMETERS (NOMENCLATURE)

M_0	Freestream Mach Number
T_0	Freestream (Inlet) Air Temperature
a_0	Speed of Sound @ Pressure Altitude in Air
g	Acceleration due to Gravity
$(U_e)_{AB}$	Exhaust Velocity at Afterburner Exit
F_S	Specific Thrust
f	Rate of Fuel Flow
TSFC	Thrust Specific Fuel Consumption
η_P	Propulsion System Efficiency
η_T	(Engine) Thermal Efficiency
η_O	(Engine) Overall Efficiency

Figure 9: Engine System Inputs/Outputs Nomenclature

We can return to our concept of M&S Theory as a "Unified Model and Experiment Design Structure", shown in Figure 10, as we now move into the "Experimental Framework Design". This framework design will show how we can incorporate these basic "time-view" experimental frames in an enhanced framework

which captures the information about the behavior of our complex system of systems, as shown in Figure 9.

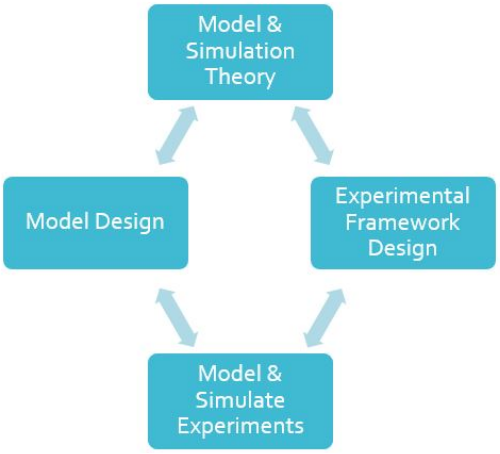


Figure 10: Unified Model — Experimental Framework

EXPERIMENTAL FRAMEWORK

Framework Objectives

To support testing of the Turbojet Engine models, we looked at developing a hierarchical test environment with an outer testing model that encapsulates for testing one coupled model. The outer testing model would provide a set of values (parameters) as input to the coupled turbojet engine model, the set of values would be propagated in one direction through the engine stages, additional values may be added at each stage as necessary, propagation terminates with output of the last stage of the engine and a final set of output values as shown in Figure 11.

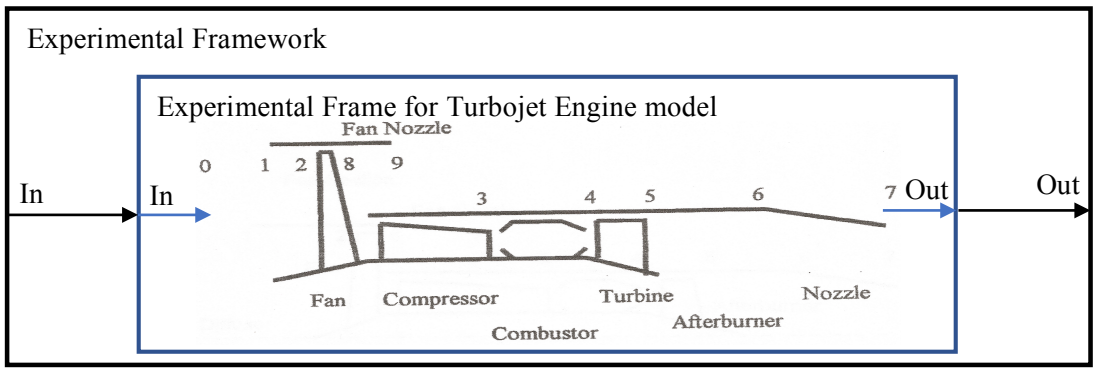


Figure 11: Experimental Framework encapsulating 1 Turbojet Engine model

We planned to develop multiple turbojet engine models to evaluate use of different gas models and engines with/without an afterburner. The testing of multiple engine models simultaneously within the same framework would simplify the task of collating results, and reduce the chances that errors would be

introduced in the process. The framework idea was extended to support multiple models whereby each model would be provided the same set of input values to operate upon as shown in Figure 12.

The framework operates as a sequence of discrete events to process input/output with the coupled model(s). The timeless nature of the encapsulating model doesn't restrict the test models from implementing their own time models so long as they are designed to being operating with the provided set of input values, and produce a set of output values in the form supported by the encapsulating framework.

Next we considered extending the framework to support running of multiple tests. Rather than develop a general test harness, we decided that there was move value in developing support for conducting Design Experiments. Design Experiment is a methodology for proposing a hypothesis, evaluating a set of features with a set of test permutations, and then analyzing the feature variance statistically to provide data for evaluating the hypothesis [7]. In support of this methodology, the framework was designed to run a suite of experiment permutations for all of the coupled models, and for each permutation run the set of values is adjusted to the design experiment factor levels.

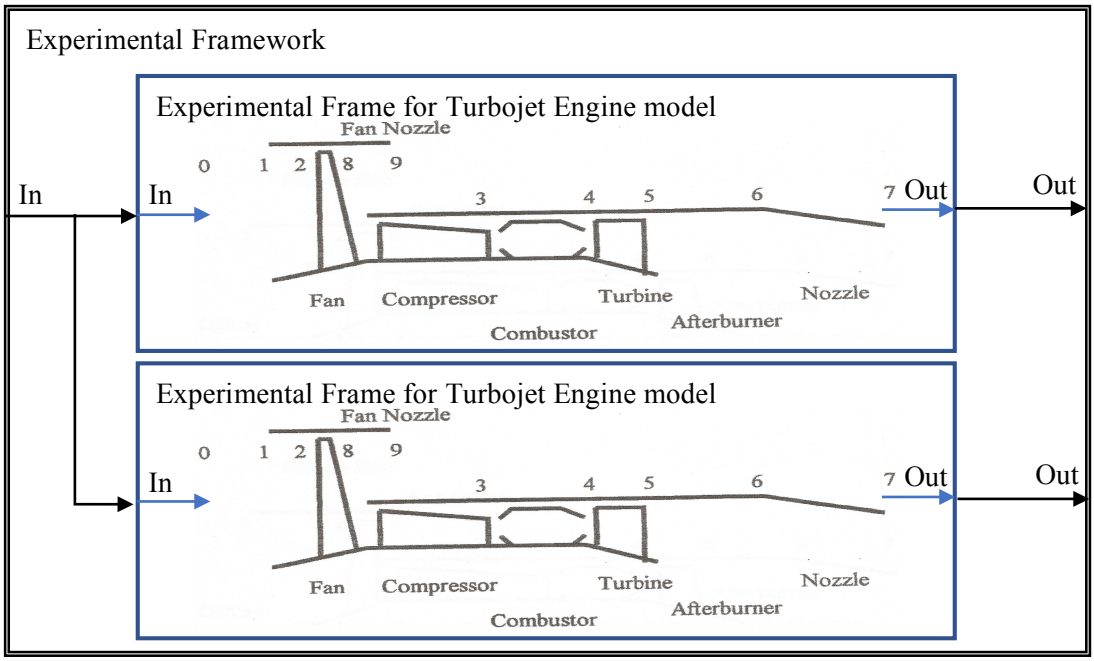


Figure 12: Experimental Framework encapsulating N Turbojet Engine models

Testing an engine given static inputs is useful, but we also want to provide the capacity to test a model for the duration of a mission. Within the domain of aircraft engine design, a mission would define specific periods of climb, cruise, descent, and other criteria to define the scope of a mission. We applied this concept to our framework first by permitting each model to determine when it had completed operation, and second

while a model's run has not completed the output of the model is fed back into the model as input. In this way the initial value set of an experimental run is morphed again and again until a mission is complete.

The key objectives for our Experimental Framework are:

1) Hierarchical test environment with one or more coupled models to be tested simultaneously.
2) Each coupled test model must support input/output of value set of the Experimental Framework.
3) The Framework operates without a time component, and the test coupled models are independent and free to operate with discrete or continuous models of time.
4) Provide support for conducting Design Experiments [7]:
    a) An input definition for an experiment. The definition defines the factors to be evaluated, the levels for each factor, and the experimental run permutations.
    b) Capacity to alter select values of the provided input for definition of experiment factors.
5) Each model self-determines when it has completed operation:
    a) The output of each coupled model is feedback as input to the same coupled model in a cycle.
    b) The feedback cycle for each model continues until the model has terminated due to a failure or the model has self-determined that it has completed its operation.
6) Provide support for recording results of test coupled models.

We choose to build upon the DEVS-Suite environment because of our common experience with the tool suite, accessibility to the suite's source code, and the suite's flexibility. We believe that the key features of the Experimental Frame are implementable for testing models developed with other modeling tools such as the commercial modeling package MathWork's Simulink.

We also investigated several features that are implementation specific for the DEVS-Suite. The most interesting of which has been an approach where the value set passed to a coupled model within the framework includes support via Java Reflection to modifying the run time variables within a model. By providing a base class for coupled models to build upon and library classes, the DEVS-Suite modeler using our framework has an abstraction that hides value encoding and decoding. This permits the modeler to focus on computation rather than infrastructure overhead.

Experimental Frame

Given the objectives outlined above we broke down the operation into a simple model composed of three components as seen in Figure 13. The first is a generator which we refer to as the Experiment Front Porch, the second is an acceptor which we refer to as the Experiment Back Porch, and the third is a transducer which we refer to as the Experiment Data Recorder. The Front Porch (generator) is responsible for initiating Design Experiment runs for all coupled models under test. The Back Porch (acceptor) is responsible for

determining if an experimental run should be recorded, which is typically done when a run has concluded. Finally, the Data Recorder (transducer) is responsible for recording output results into a format that will facilitate analysis. (See Appendix E for source of the Front Porch / Back Porch nomenclature.)

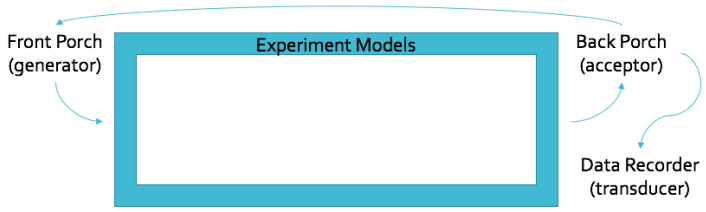


Figure 13: Experimental Frame for the Experimental Framework

The Experimental Framework implementation is divided into two layers. The first layer is the experiment base layer that implements basic model semantics within the DEVS-Suite to react to external events, process internal events, and output messages. Built upon this is the framework layer that instantiates the functionality of the Front Porch, Back Porch, and Data Recorder. A summary view of the base layer is shown in Figure 14, and a detailed view of the class models for the base layer and the framework layer may be found in Appendix C.

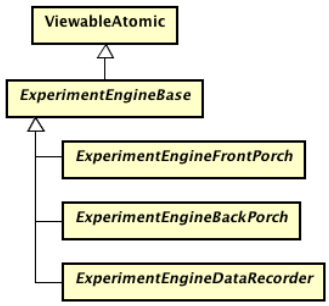


Figure 14: Class ExperimentEngineBase

The following sections will look at the three components of the framework layer in more detail.

Experiment Front Porch

The key objectives for our Experimental Framework that are addressed by the Front Porch are:

1) Hierarchical test environment with one or more coupled models to be tested simultaneously.
2) Each coupled test model must support input/output of value set of the Experimental Framework.
3) The Framework operates without a time component, and the test coupled models are independent and free to operate with discrete or continuous models of time.
4) Provide support for conducting Design Experiments:
    a) An input definition for an experiment. The definition defines the factors to be evaluated, the levels for each factor, and the experimental run permutations.

b) Capacity to alter select values of the provided input for definition of experiment factors.

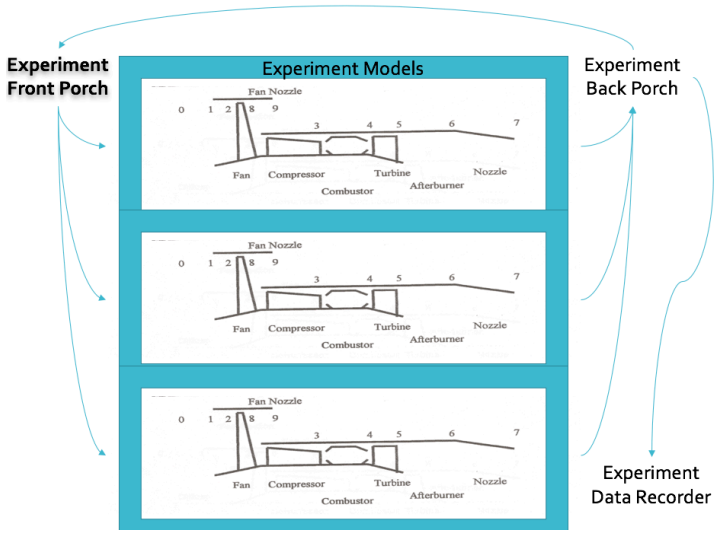


Figure 15: Experiment Front Porch

The Front Porch is roughly responsible for starting the Design Experiment for each of the models under test, to keep a test run's output-to-input looping around until complete, and starting the run for a model until each experiment has completed for all of the models.

Hierarchical test environment

The Front Porch runs one experiment at a time. That one experiment will have one or more test runs which are applied to one or more coupled models simultaneously. The rate at which each model operates through a test and is ready for the next run of the experiment is independent of the other models under test.

The testing multiple related models in parallel may be utilized for a variety of goals. The most basic example would be comparison of a current working model against a newer version that should offer improvement over its predecessor. Providing a hierarchical test environment that evaluates both versions helps ensure that the output results are collated without introduction of additional human error that is often introduced when a sequence of tests need to be executed and compared. For testing of a turbojet engine there are a myriad of uses such as comparing different gas models, with and without an afterburner, or alternate sequencing of engine stages.

Test model support for input/output of value set

A sample base class is provided to a test model to build upon that support the input and output of value sets. This is the platform upon which the Turbojet Engine modes were built. This is elaborated upon on page 27 in "Experiment Model Support."

Timeless operation

The Front Porch, as well as the Back Porch and Data Recorder, operate with a simple event based model. State for these model components moves ahead based on the propagation of value sets through the model. This timeless operation is atypical of models built upon the DEVS-Suite. The simplicity of the outer model aids in the blending of the nested coupled test models that may implement a time model independent of the encapsulating Experimental Framework.

Support for conducting Design Experiments

After developing an hypothesis, selecting factors, and designing an experiment the factor levels and runs may be entered into a JSON file. The file provides a Design Experiment definition that drives the operation of the Front Porch for the execution of one complete experiment for all of the models under test. A sample file is provided in Figure 16.

```
{
  "<factors>" : [
    {"Compressor Pressure Ratio" :
      ["pi_c", {"Low" : "13.5", "Med" : "24.25", "Med2" :
"27.045",
                "Hi" : "35.0"}]},
    {"Bypass Ratio" :
      ["alpha", {"Low" : "0.0", "Med" : "1.0", "Hi" : "2.0"}]},
    {"Turbine Inlet Temperature" :
      ["Tt4", {"Low" : "1200", "Med" : "2400", "Med2" : "1848",
               "Hi" : "3600"}]},
    {"Afterburner State" :
      ["afterburnerOn", {"Off" : "false", "On" : "true"}]}
  ],
  "<runs>" : [
    {"///+" : ["Med",  "Med",  "Med",  "On"]},
    {"-/+-" : ["Low",  "Med",  "Hi",  "Off"]},
    {"/-/-" : ["Med",  "Low",  "Med", "Off"]},
    {"-+--" : ["Low",  "Hi",  "Low", "Off"]},
    {"--++" : ["Low",  "Low", "Hi",  "On"]},
    {"/+/-" : ["Med",  "Hi", "Med",  "Off"]},
    {"-+++" : ["Low",  "Hi", "Hi", "On"]},
    {"+++-" : ["Hi",  "Hi", "Hi", "Off"]},
    {"/+-+" : ["Med", "Hi",  "Low",  "On"]},
    {"+/--" : ["Hi", "Med",  "Low",  "Off"]},
    {"----" : ["Low", "Low",  "Low", "On"]},
    {"-//+" : ["Low", "Med",  "Med2", "On"]},
    {"++/+" : ["Hi", "Hi", "Med",  "On"]},
    {"+--+" : ["Hi", "Low", "Low",  "On"]},
    {"//++" : ["Med2", "Med", "Hi", "On"]},
    {"+-+-" : ["Hi", "Low", "Hi", "Off"]}
```

Figure 16: Sample Design Experiment JSON File

The factors to be tested for a specific experiment are values (parameters) for the test models. Each time a new experimental run is prepared for a model, a clean input value set is loaded and the set of factors for the newest experiment are applied. This mutated value set with the features updated is then transmitted to the test model.

Experiment Back Porch

The key objectives for our Experimental Framework that are addressed by the Back Porch are:

5) Each model self-determines when it has completed operation:
    1. The output of each coupled model is feedback as input to the same coupled model in a cycle.
    2. The feedback cycle for each model continues until the model has terminated due to a failure or the model has self-determined that it has completed its operation

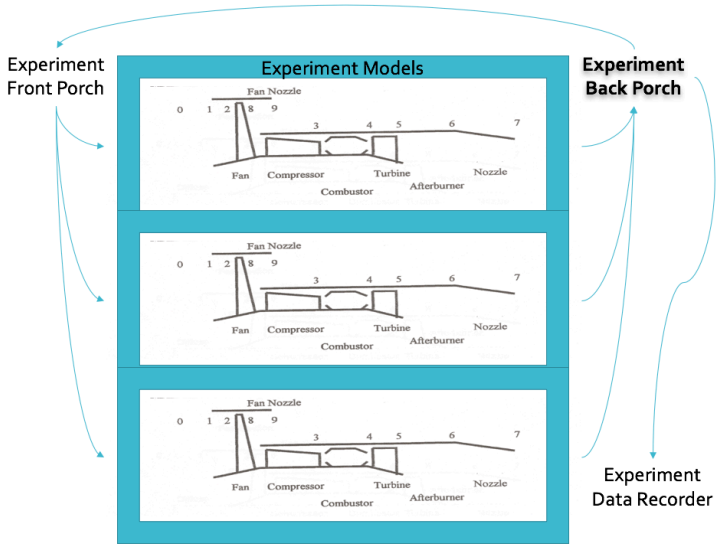


Figure 17: Experiment Back Porch

The Back Porch's role is to evaluate if a model's experimental run has completed or terminated, and send the output to both the Front Porch and the Data Recorder.

Self-determine Completion

There are three defined ways that a model's experimental run may end an infinite cycle of input-to-output. First, the model may announce while executing that it has completed its run. This could be on the initial pass or for example after a simulated mission has completed. Second, the model may announce while executing that it has terminated its run. An example case can be made for the turbojet engine model to terminate an internal pressure calculation exceeds that permitted for current material strength. Finally, there is support for a function to be registered with the Back Porch model. This auxiliary function is queried to determine if a model has completed its run.

Experiment Data Recorder

The key objective for our Experimental Framework that is addressed by the Data Recorder is:

6) Provide support for recording results of test coupled models.

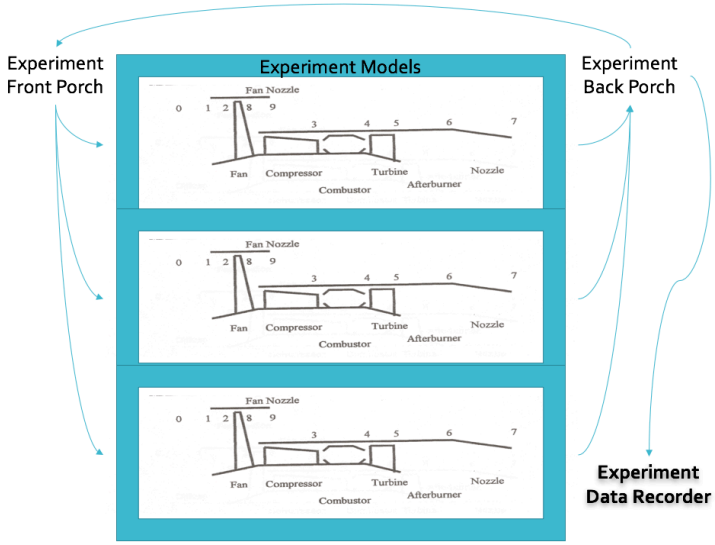


Figure 18: Experiment Data Recorder

The Data Recorder's role is to record the output values of the test coupled models. The fundamental data to be recorded for each model of each experimental run is the final result set. Some experimentation may require snapshot records of interim execution passes. For example if a turbojet engine follows a flight plan then data for multiple time slices or elevation levels may need to be required.

Presently there is support for recording the output values in two formats. The first format is a Text dump of the value set with group, units, and summary comments. The second format is Comma Separated Value (CSV) dump that contains a header row and the raw value set.

Output report as Text

The value definition within the Experimental Framework provides a rich set of qualities for each value. These qualities are utilized to their fullest in output of the text report. A single file contains all the reports for all model runs of an experiment. Each model's output begins with a line describing the experiment state, followed by the set of values for the model. First are values provided as input to the experiment models. This is followed, in order, by values that have been added by a specific coupled model. For the example subset of values displayed in Figure 19, the Mission Setup, Diffuser, Compressor, etc. are model components of a turbo jet engine.

```
Model = A; Run = 1; Pass = 1; Terminated = false; Completed = true
Initial State
  Tzero: 411.8 [degR]                                    Inlet temperature
  Pzero: 629.6 [lbf/ft^2]                                Inlet pressure
  Mzero: 0.9                                             Subsonic flight Mach
  gamma: 1.4 [Cp/Cv]
  Rair: 53.35 [ft-lb/degR-lbm]                           Specific gas cnst. dry air
  gc: 32.174 [lbm-ft/s^2]                                gravity cnst
  Cp: 0.246 [Btu/lbm-degR]                               Specific heat of dry air
  pi_c: 17.8 [Pt3/Pt2]                                   Compressor Pressure Ratio
  mdot_zero: 170.0 [lbm/s]                               Mass flow rate of air
  hpr: 18400.0 [Btu/lbm]                                 Specified enthalpy of fuel
  Tt4: 2000.0 [degR]                                     Turbine Inlet Temperature
Mission Setup
  azero: 31.086866724111264 [ft/s]                       Speed of sound
Diffuser
  Tt2: 478.5116                                          Stagnation Temp. @ station 2
  Pt2: 31.638194372671034                                Stagnation Press. @ station 2
  tao_r: 1.1619256134033202                              Temp. Ratio @ station 2
  pi_r: 1.691303112850931                                Press. Ratio @ station 2
Compressor
  tao_c: 2.2764748558864896                              Temp. Ratio @ station 3
  w_c: 150.2587743099011 [Btu/lbm]                       Specific work, compressor
  Wdot_c: 36123.01916080482 [hp]                         Compressor Work
  Pt3: 31.59874561979826                                 Stagnation Pressure @ stage 3
  psi: 3.9108323599551693 [psi]                          Stagnation Pressure @ stage 3 (in psi)
  Tt3: 1089.3196256500134                                Stagnation Temp. @ stage 3
Combustor
  tao_lambda: 4.856726566294317                         Temp. Ratio @ station 4 inlet (station 3 outlet)
  Pt4: 31.59874561979826 [Btu/lbm]                       Static Pressure @ station 4 inlet Pt3=Pt4
  pi_b: 1.0                                              Pressure ratio at burner (combustor)
  f: 0.012176332968854574                                fuel flow (ratio)
  tao_b: 1.8361259559085197                              Temp. ratio at burner (combustor)
```

Figure 19: Sample of output report as Text

Output report as CSV

The values for all model runs of an experiment are written as a Comma Separated Value text file and is easily loaded into a spreadsheet tool for analysis. The value definition within the Experimental Framework also provides a way for individual values to omitted from the output. When a complex coupled model may contain hundreds of values, the capacity to record only a subset of them is useful to reduce an overload of data. Each model's output begins with columns describing the experiment state, followed by columns for values for the model being tested. For the example subset of values displayed in Figure 20, the rows have been sorted by Model, only a few of the experiment runs are displayed, and only a few of the model input value columns are displayed.

Model	Run	Pass	Terminated	Completed	Tzero	Pzero	Mzero	gamma	Rair	gc	Cp	pi_c
A	1	1	FALSE	TRUE	411.8	629.6	0.9	1.4	53.35	32.174	0.246	17.8
A	2	1	FALSE	TRUE	411.8	629.6	0.9	1.4	53.35	32.174	0.246	17.8
A	3	1	FALSE	TRUE	411.8	629.6	0.9	1.4	53.35	32.174	0.246	17.8
A	4	1	FALSE	TRUE	411.8	629.6	0.9	1.4	53.35	32.174	0.246	17.8
A	5	1	FALSE	TRUE	411.8	629.6	0.7	1.4	53.35	32.174	0.246	17.8
A	6	1	FALSE	TRUE	411.8	629.6	0.7	1.4	53.35	32.174	0.246	17.8
B	1	1	FALSE	TRUE	30	411.8	629.6	0.9	1.4	53.35	32.174	0.246
B	2	1	FALSE	TRUE	30	411.8	629.6	0.9	1.4	53.35	32.174	0.246

Figure 20: Sample of output report as CSV

Experiment Model Support

To support the development of models that utilize the Experimental Framework, a base class and interface are provided (see Figure 21). The base class interacts with the DEVS-Suite to process external events, internal events, and output. This class is strictly limited in implementation to the input/output of experiment value sets. If your model needs additional inputs/outputs then the methods may be overload, or the source may be used as a reference. This is the foundation utilized by the Turbojet Engine models.

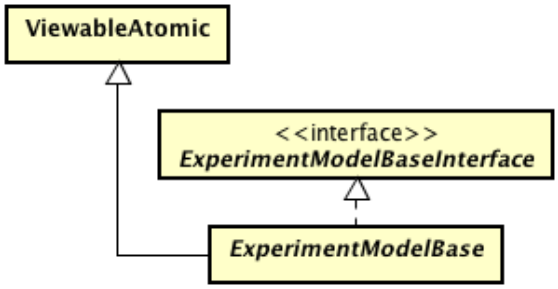


Figure 21: Class ExperimentModelBase

UML details for the provided base class and interface is in Figure 22. The base class interacts directly with the DEVS-Suite, and the interface provides two functions that are to be implemented by a derived class.

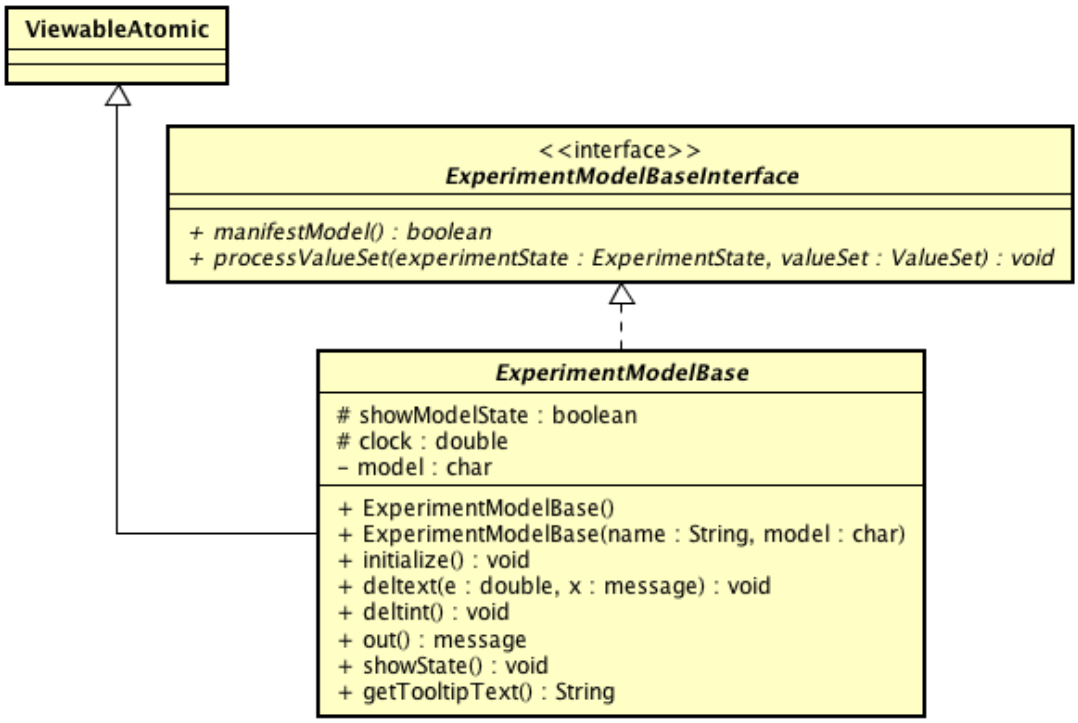


Figure 22: Class detail for ExperimentModelBase

Function required of derived class: `manifestModel()` — This function is provided so that a model under test may have individual components decide at runtime if they are to be instantiated.

Function required of derived class: `processValueSet()` — This function is provided to pass the current experiment's run state and the value set to a model component. The value set contains all of the input data values to be manipulated (see section Value Management on page 30 for more details).

Experiment Support Functionality

Support functionality was developed to aid configuration and implementation of experiment models. Here is a brief introduction to the functionally.

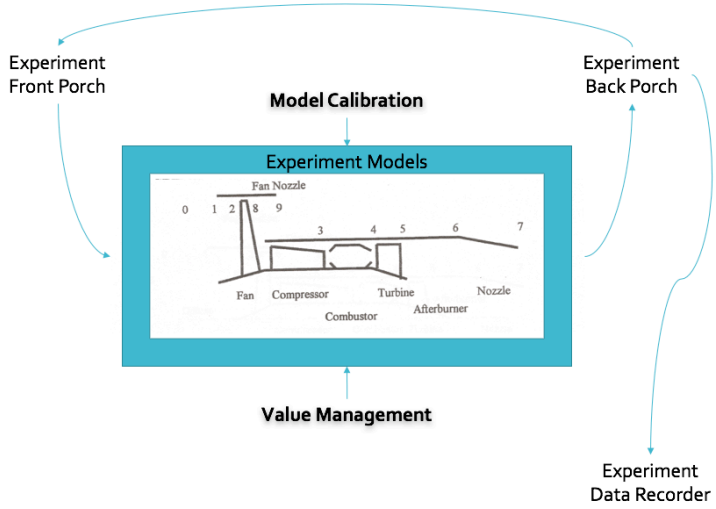


Figure 23: Experiment Support Functionality

Property Settings

To provide external control of properties that affect the Experimental Framework, we provide a class, and its Singleton companion, are used to read as property file at startup and provide interfaces for looking up a value from a name. All of the Java base types are supported, as well as hexadecimal and octal representations for the various integer forms.

```
# Settings.properties

# Number of concurrent experiment models to run. Range from 1 to 26.
ExperimentEngine_ModelCount = 2

#CalibrationPath=.\xxx\yyy
#CalibrationFile=MyCalibration.json
CalibrationDebug=true
```

Unit to Type Mapping

We experimented with the notion that the runtime platform may want to enforce strong type mapping of variable names. To support this we provide a class that supports mapping from a unit of measure to a

specific Java type. For example, a naming convention may use “a” for acceleration and “Da” for the 1st derivate of acceleration. We want to enforce that there two units labels are always represented using Double. For this to be used, a platform would need to enforce a specific variable naming convention so that the units may be extracted from the variable name. The following is a sample input file that defines the units that may be used in variable names for specific Java types, and in this example “a” and “Da” are valid units that are mapped to the type Double.

```
{
  "boolean" : ["b"],
  "char"    : [],
  "byte"    : [],
  "short"   : [],
  "int"     : ["Cnt","ms"],
  "long"    : [],
  "float"   : [],
  "double"  : ["a", "Da"],
  "String"  : ["s"]
}
```

Calibration

A common practice for engine controllers is to provide a way for constant element and arrays that define the engine behavior at runtime to be tunable without altering the code. For engine controllers developed in languages like C/C++, the alterable constants are stored at link time into a read-only memory segment that can updated under special conditions such as during vehicle servicing.

Our Java based calibration feature uses a configuration file for each test model. As the Experimental Framework starts, each test model loads its calibrations, Then within the model, the calibration object may be invoked to overwrite constants with the externally defined calibration values using Java Reflection. The calibration feature also uses a structured variable naming convention. Here is an example calibration input file, followed by a code fragment showing the use of a calibration:

```
{
  "KeTJET_Bool_BlackBox" : "true",
  "KeTJET_Bool_Afterburner" : "false",
  "KeTJET_Cnt_SimViewRows" : "2",
  "KeTJET_Str_FluidModel" : "Isentropic/Static (subsonic)"
}
```

```
public class Afterburner extends ... implements ... {

  private final String KeTJET_Str_FluidModel;
  private final boolean KeTJET_Bool_Afterburner;

  public Afterburner(. . .) {
    KeTJET_Str_FluidModel = "undefined";
    KeTJET_Bool_Afterburner = false;
  }

  public void process(final ExperimentState experimentState, final ValueSet valueSet) {
    switch (KeTJET_Str_FluidModel) {
    case "Isentropic/Static (subsonic)":
      IsentropicStaticSubsonic(experimentState, valueSet);
      break;
```

```
    case "Isentropic/Static (supersonic)":
      IsentropicStaticSupersonic(experimentState, valueSet);
      break;
    case "Polytropic/Static (supersonic)":
      PolytropicStaticSupersonic(experimentState, valueSet);
      break;
    default:
      System.out.println(name + "Undefined fluid model '" + KeTJET_Str_FluidModel + "'");
      break;
    }
  }
```

Value Management

Given as a starting point that a Turbojet Engine would be a model with many input values (parameters) and many output values we needed to consider how we wanted to model this data movement within the DEVS-Suite. We decided to try the idea of passing a set of values as a single message between model components. That would simplify the message processing portion of our models, simplify the external/internal event processing, and provide an interesting area of functionality exploration.

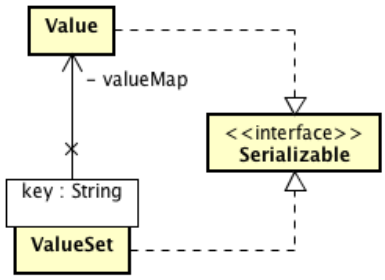


Figure 24: Class UML for Value and ValueSet

We defined a "Value" class to contain a generic Java Object. This object may be cast as a Boolean or Double. A value object holds the value's type, the current parameter, and properties that describe the unit of measure and other qualities. A class to contain a set of these values is used to pass about the set (Figure 24). Most important is functionality to take the current values from the set and overload class variables with a method call, and to perform the reverse. For example, the model user receives a value-set as input. They call a method to "pop" the values out of the value-set and into local class variables. Then calculations are performed and when complete a method is called "push" the values back into the value-set.

For the Experimental Framework, the messages passed between the Front Porch, Back Porch, Data Recorder and the coupled test models is a pair of two structures. The 1st part of the pair is an experiment state object for tracking the model and run that this message is associated with. The 2nd part of the pair is a value-set as described above.

The following is an example value-set input file, followed by a code fragment showing the use of a value-set as input to a model at runtime. In this example the model adds three new values to the output value-set:

```
{
  "Isentropic/Static/Subsonic" : [
    { "Initial State" : [
      {"Tzero" : ["double", "411.8", "[degR]", "Inlet temperature", "", true]},
      {"Pzero" : ["double", "629.6", "[lbf/ft^2]", "Inlet pressure", "", true]},
      {"Mzero" : ["double", "0.8", "", "Subsonic flight Mach", "", true]},
      {"gamma" : ["double", "1.4", "[Cp/Cv]", "", "", true]},
      {"Rair" : ["double", "53.35", "[ft-lb/degR-lbm]", "Specific gas cnst. dry air", "", true]},
      {"gc" : ["double", "32.174", "[lbm-ft/s^2]", "gravity cnst", "", true]},
      {"Cp" : ["double", "0.246", "[Btu/lbm-degR]", "Specific heat of dry air", "", true]},
      {"pi_c" : ["double", "13.5", "[Pt3/Pt2]", "Compressor Pressure Ratio", "", true]},
      {"mdot_zero" : ["double", "170", "[lbm/s]", "Mass flow rate of air", "", true]},
      {"hpr" : ["double", "18400.0", "[Btu/lbm]", "Specified enthalpy of fuel", "", true]},
      {"Tt4" : ["double", "1669.670", "[degR]", "Turbine Inlet Temperature", "", true]}
    ]}
  ]
}
```

```
public class ExampleTurbojetClass extends ... implements ... {

  private double Tzero, gamma_c, gamma_t, Rair, gc, Cp_c, Cp_t, azero, Rair_c, Rair_t;

  public void process(final ExperimentState experimentState, final ValueSet valueSet) {
    valueSet.valuePop(this);

    // [ft-lb/degR-lbm] Specific gas cnst. dry air (cool)
    Rair_c = ((gamma_c-1)/gamma_c)*Cp_c*778.17;

    // [ft-lb/degR-lbm] Specific gas cnst. dry air (heated)
    Rair_t = ((gamma_t-1)/gamma_t)*Cp_t*778.17;

    // [ft/s] speed of sound
    azero = Math.sqrt(gamma_c*Rair_c*gc*Tzero);

    if (experimentState.isFirstPass()) {
      valueSet.addValue(name, "Rair_c", Value.theType.eDouble, Rair_c, "[ft-lb/degR-lbm]",
                        "Specific gas cnst. dry air (cool)", "", true);
      valueSet.addValue(name, "Rair_t", Value.theType.eDouble, Rair_t, "[ft-lb/degR-lbm]",
                        "Specific gas cnst. dry air (heated)", "", true);
      valueSet.addValue(name, "azero", Value.theType.eDouble, azero, "[ft/s]", "Speed of sound", "", true);
    }
    valueSet.valuePush(this);
  }
```

Finally, we will explore the unified results of our designs and conclude the "Unified Model and Experiment Design Structure", shown in Figure 25.

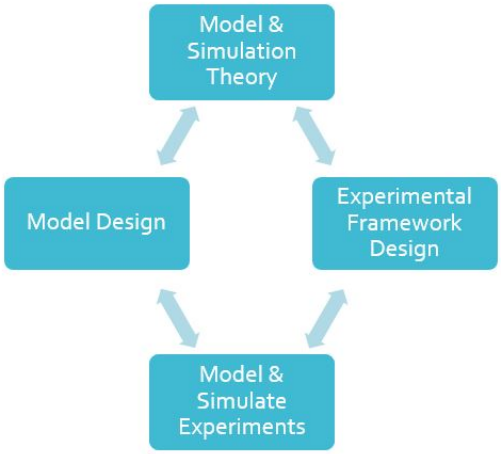


Figure 25: Unified Model — Model & Simulate Experiments

MODEL AND SIMULATE EXPERIMENTS

Now that we have designed both the Model (Turbojet Engine Model System and Components), and the Experimental Framework, (as in the previous section), the stage is set to run actual experiments relating more specifically to the Turbojet Engine research we would like to perform. But before running the designed set of experiments, it is important to verify and validate (V&V) the product. We would like to verify that we built the product right, and that we built the right product.

Verification &Validation Testing

Verification

Our verification testing was performed with a 3-phase testing strategy. We are able to verify that the output calculations from three different sources match with only a minimal percent of error difference. We chose to design test sets in both the subsonic and supersonic flight regime, with both afterburner, and non-afterburner mode on, and with different flight altitudes. These test sets were run against three model calculation types:

1. Mathematical calculations
2. AIAA AEDSys software application calculations
3. DEVS-Suite Experimental Framework

The results from the test cases show that the models matched up with only a small amount of error difference. In all cases, the mathematical calculations matched those of our DEVS-Suite Experimental Framework, and so we compare the DEVS-Suite and AEDSys together. We can look at some of the results more closely and analyze them for verification.

Isentropic/Subsonic Test Case

In this test run, we designed the test to model subsonic flight. This was our simplest baseline test to verify the Isentropic/Subsonic model. We can view the first set of results from this test in Figure 26,. The results show an error ranging from only 1% to 6% for the measured efficiencies between the two models. Similar results for the other measured output variables were also viewed and thus verified our Isentropic/Subsonic model. (Another output result of the measured TSFC is shown in Figure 27.)

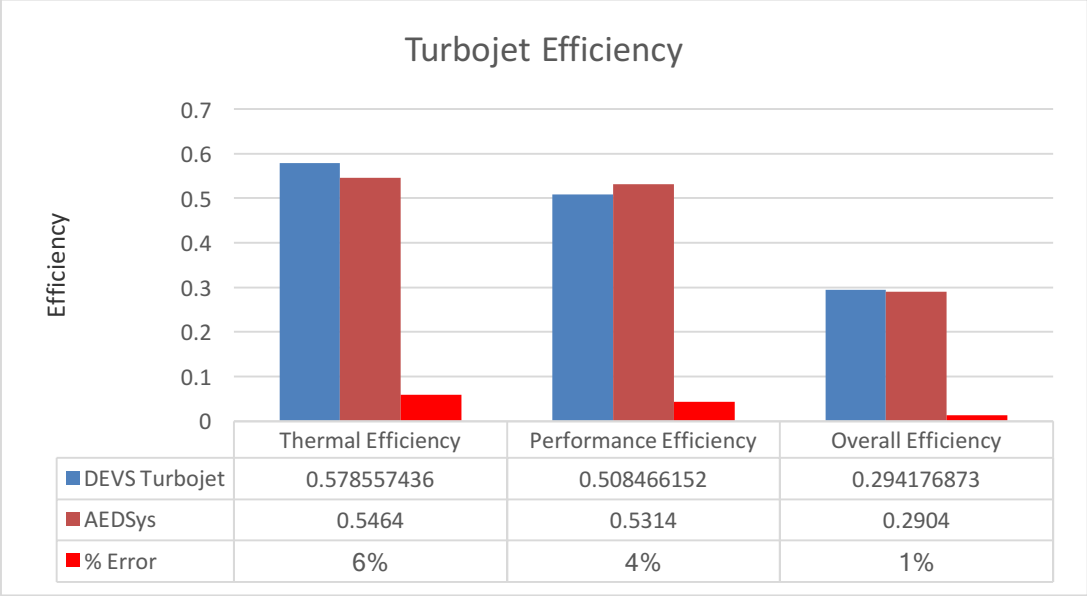


Figure 26: Isentropic/Subsonic Test Results – Efficiency

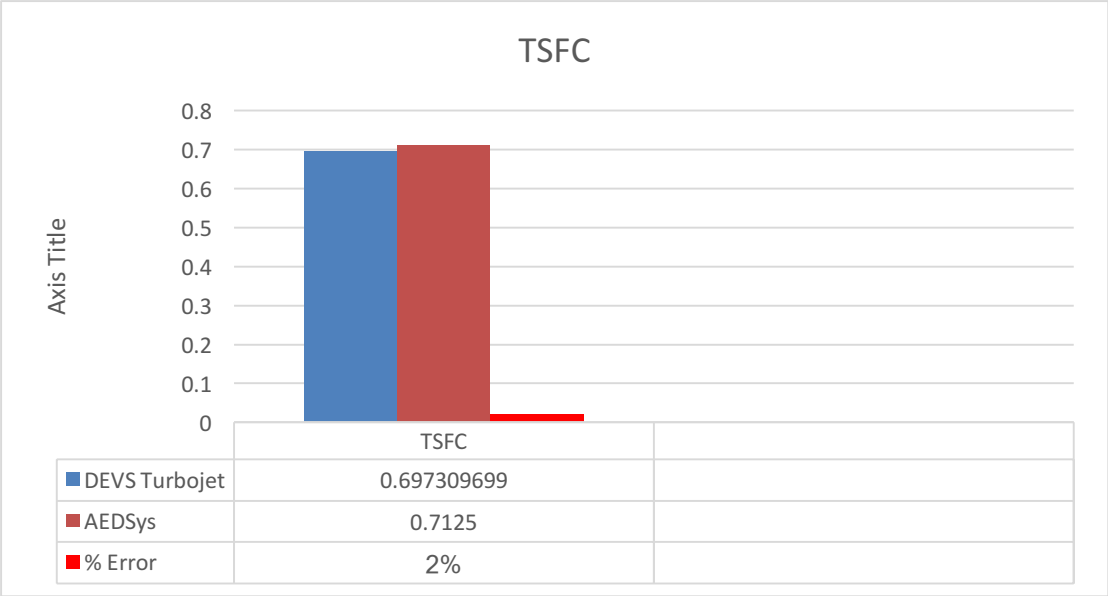


Figure 27: Isentropic/Subsonic Test Results - TSFC

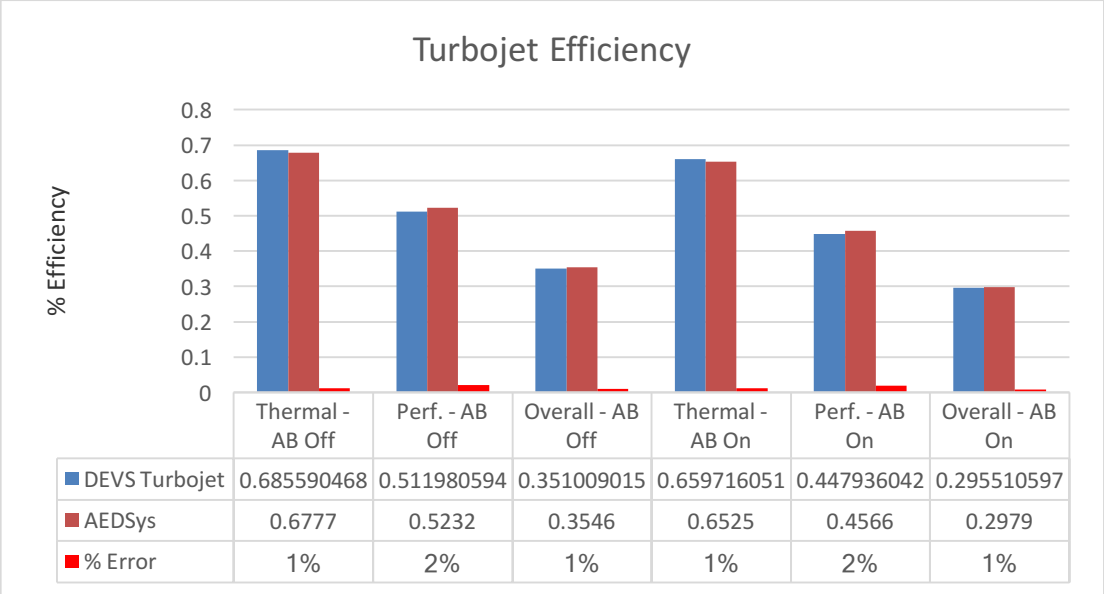
Isentropic/Supersonic Test Case

The next test case modeled a supersonic flight with both afterburner on/off options. Again the results between AEDSys and the DEVS-Suite Experimental Framework design fared well with each other. From the results, we show an error of difference of only 2%, and in most cases only a 1% error. A graph of the

turbojet efficiencies is shown in Figure 28. Again, similar results for the other measured variables showed promising results and thus verified our Isentropic/Supersonic model.

Polytropic/Supersonic Test Case

The final set of test cases verified our Polytropic/Supersonic models. This would be the models mostly to be used in the design of experiments research runs, and depended upon good verification results. From the graph in Figure 29, the results show an error of 1% in most cases, with a slightly higher 4% error in one case, and a 13% error as well in one case. The results show, however, that the difference between the designed DEVS-Suite Experimental Framework are slightly more conservative in the calculation of the efficiency of the turbojet engine versus the prediction of the AEDSys model. For our research purposes, we are attempting to model a realistic engine, and further optimize where we can. We would like to be as exact as possible, however, if we are to err on one side or the other, it would be beneficial to be more conservative in our calculations. This would allow for the benefit of having even better efficiency results show up in a real (physical) engine test run. If we were to calculate results showing unreachable levels, we might waste valuable resources attempting to gain these unrealistic efficiencies. It is then acceptable to have the 13% error on a conservative side for our framework design. Again, similar results from other measured variables showed promising results and verified our Polytropic/Supersonic model.



	Thermal - AB Off	Perf. - AB Off	Overall - AB Off	Thermal - AB On	Perf. - AB On	Overall - AB On
DEVS Turbojet	0.685590468	0.511980594	0.351009015	0.659716051	0.447936042	0.295510597
AEDSys	0.6777	0.5232	0.3546	0.6525	0.4566	0.2979
% Error	1%	2%	1%	1%	2%	1%

Figure 28: Isentropic/Supersonic Test Results – Efficiency

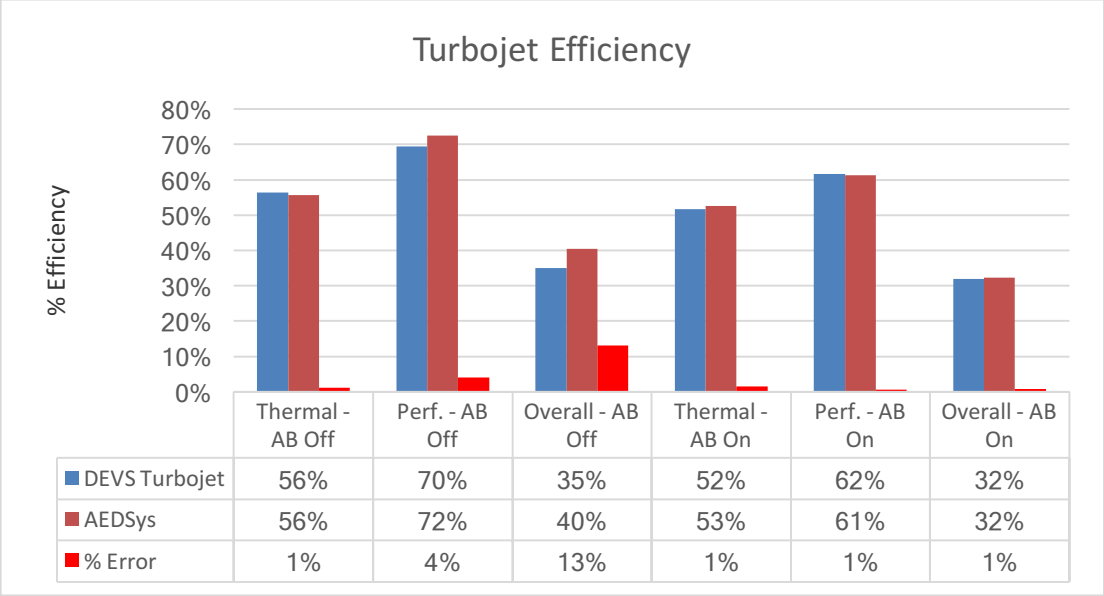


Figure 29: Polytropic/Supersonic Test Results - Efficiency

Validation

To perform validation, we return to our requirements and desired specifications for performance, configuration, and usability. We are looking for a framework and model and simulation design which is built with a good design, a good architecture, able to handle many models at once, different configurations of models, and gives consistent output.

For these tests, we incorporated three different model configurations, and a set of over 50 runs at a time with multiple variable alterations on all three model configurations. This system testing proved to be quite capable, even more so than expected. The execution time for this stress test took less than 10 seconds to complete, and the output data was verified to be correct for all models and all variations. Further, with the enhanced abilities of the "calibration feature", the ease of setting up experiments took a short learning curve, and less than a few minutes to create. The framework offers the independence of models from the experimental frame and this decoupling allows for the model types to be modified, swapped in and out, and systems to contain different amounts of components and different component types altogether.

Overall, the system exceeded the expectations of what was required and the capabilities of this framework for our research purposes proved to be exceptional. An example of the data output from an experimental run can be seen in Figure 30. As one can see, the data output is well organized, readable, and consistent with the data from which the aerospace community of research is accustomed to.

```
Model = A; Run = 1; Pass = 1.6; Terminated = false; Completed = true
Initial State
  Tzero: 411.8 [degR]                                   Inlet temperature
  Pzero: 629.6 [lbf/ft^2]                               Inlet pressure
  Mzero: 0.9                                            Subsonic flight Mach
  gamma: 1.4 [Cp/Cv]
  Rair: 53.35 [ft-lb/degR-lbm]                          Specific gas cnst. dry air
  gc: 32.174 [lbm-ft/s^2]                               gravity cnst
  Cp: 0.246 [Btu/lbm-degR]                              Specific heat of dry air
  pi_c: 17.8 [Pt3/Pt2]                                  Compressor Pressure Ratio
  mdot_zero: 170.0 [lbm/s]                              Mass flow rate of air
  hpr: 18400.0 [Btu/lbm]                                Specified enthalpy of fuel
  Tt4: 2000.0 [degR]                                    Turbine Inlet Temperature
Mission Setup
  azero: 994.7797351715604 [ft/s]                       Speed of sound
Diffuser
  Tt2: 478.5116                                         Stagnation Temp. @ station 2
  Pt2: 1064.8444398509462                               Stagnation Press. @ station 2
  tao_r: 1.162                                          Temp. Ratio @ station 2
  pi_r: 1.691303112850931                               Press. Ratio @ station 2
Compressor
  tao_c: 2.2764748558864896                             Temp. Ratio @ station 3
  w_c: 150.25877430990332 [Btu/lbm]                     Specific work, compressor
  Wdot_c: 36123.01916080482 [hp]                        Compressor Work
  Pt3: 18954.231029346844                               Stagnation Pressure @ stage 3
  psi: 131.62660437046418 [psi]                         Stagnation Pressure @ stage 3 (in psi)
  Tt3: 1089.3196256500134                               Stagnation Temp. @ stage 3
Combustor
  tao_lambda: 4.856726566294317                         Temp. Ratio @ station 4 inlet (station 3 outlet)
  Pt4: 18954.231029346844 [Btu/lbm]                     Static Pressure @ station 4 inlet Pt3=Pt4
  pi_b: 1.0                                             Pressure ratio at burner (combustor)
  f: 0.01217540065707047                                fuel flow (ratio)
  tao_b: 1.8360084156260104                             Temp. ratio at burner (combustor)
Turbine
  w_t: -150.25877430990332 [Btu/lbm]                    (turbine work is = compressor work)
  Wdot_t: -36123.01916080482 [hp]                       Turbine Specific Work
  tao_t: 0.6945959871749933                             Temperature ratio at turbine
  pi_t: 0.2792948610700821                              Pressure ratio at turbine
  Tt5: 1389.1919743499866 [degR]                        Static Temp. @ Turbine exit/Nozzle entrance
  Pt5: 5293.819322031666 [lbf/ft^2]                     Static Press. @ Turbine exit/Nozzle entrance
Nozzle
  P7: 629.6 [lbf/ft^2]
  T7: 756.0682655547911 [degR]
  Pt7: 5293.819322031666 [lbf/ft^2]
  Tt7: 1389.1919743499866 [degR]
  pi_n: 1.0                                             Pressure ratio at Nozzle
  tao_n: 1.0                                            Temp. ratio at Nozzle
  M_7: 2.0462033236847432                               Mach number at Nozzle Exit
  U7/a7: 2.0462033236847432                             Velocity ratio at Nozzle exit
  U7/a0: 2.772592981423893                              Velocity ratio over Ambient speed of sound
  F_s: 57.89822683362291 [lbf/(lbm/s)]                  Specific Thrust
  F_7: 9842.698561715893 [lbf]                          Thrust
  TSFC: 0.7570429141363567 [(lbm/hr)/lbf] or [1/hr]     Thrust Specific Fuel Consumption (TSFC)
  eta_t: 0.6219658672226045                             Thermal Efficiency of Turbojet Engine
  eta_p: 0.4901169307637586                             Performance Efficiency of Turbojet Engine
  eta_o: 0.30483600188296234                            Overall Turbojet Engine Efficiency
```

Figure 30: Example DEVS-Suite Experimental Framework Output Data

Design of Experiments (DOE)

After validation of the product, the stage is set to continue the goal of research. This sections explains the "Design of Experiments" process taken to model, simulate, gather and analyze data, and make educated conclusions about the behavior of the turbojet engine during our required supersonic flight regime. The goal is to offer insight as to the possible opportunities for optimized turbojet engine parameter settings for future, long-duration, supersonic flights across land for commercial passengers. Another goal is to provide

the framework as a platform for future researchers (including the Authors) and requires offering them a flexible, systematic, and repeatable process for research.

We begin our research and development (R&D) problem by determining what questions are to be answered. The questions gather around the performance of the turbojet engine and what variables we presume have a significant effect on the operation behavior. By analyzing the mathematical relationships within the model, including the functions, and desired outputs, we can come up with some example questions to be answered:

- What effect does the pressure compression ratio in the compressor stage have on the performance of the turbojet engine, during long-duration, supersonic flight?
- Can we eliminate the use of a Hi-Bypass (ratio) fan, in exchange for either a Low-Bypass (ratio) or (straight) turbojet engine, for additional propulsion, and still meet thrust and fuel requirements for long-duration supersonic flight? (without the use of afterburner)
- What effect does decreasing the limitations/constraints of the Turbine Inlet temperature offer for the turbojet performance during long-duration supersonic flight? (i.e. Increasing the allowable temperature by use of new, high tolerance materials or designs)
- If the above parameters show a significant effect on the turbojet performance, is there a combination of pressure compression ratio, (low) by-pass ratio, and turbine inlet temperature that would optimize turbojet performance during long-duration, supersonic flight?

We proceed to attempt to address these questions with a Design of Experiments approach with a strategy as follows:

- Phase I - Initialization, Verification & Validation of Model (Noted in previous section)
- Phase II – Design of Experiments for Analysis
- Phase III – Design of Experiments for Optimization

Design of Experiments for Analysis

The DOE offers a systematic approach for experimentation. The fact that we have several variables to inspect, each at multiple levels, creates a very large range of experimental combinations which would be exhaustive to analyze. We employ the help of a fractional factorial design to initially narrow down the possibilities of interaction effects from the different variables of interest. We should note that the framework is well capable of executing many more experiments than is necessary at this point, however, in many research cases, the design of the experiments themselves can prove to be a non-trivial task and it is most common to employ fractional factorial designs to discover whether continuing research is promising, or if the designs should be changed at an early stage. We take this approach to offer a more realistic experimentation, more commonly familiar with the community.

The fractional factorial experiment design is first though out and detailed as follows:

Fractional Factorial Experiment:

- 4 Factors
  - Pressure Compression Ratio
  - By Pass Ratio (BPR)
  - Turbine Inlet Temperature
  - Afterburner
- 2 Levels, (Each Factor)
  - Pressure Compression Ratio (Low, Hi)
  - BPR (Low, Hi)
  - Turbine Inlet Temperature (Low, Hi)
  - Afterburner (On, Off)

The next step is to design the experiment runs. We would like to eliminate any bias in our design. With the use of JMP Statistical Software, we enter our factors, levels, and outputs to be measured, and choose a randomized design. JMP takes the information, and from all the possible combinations, outputs a design run of the desired amount of experimental runs. An easy to read experiment table can be shown below in Table 4.

Table 4: Design of Experiments — Turbojet Experiment Set 01

Turbojet Experiment Set 01: 16 runs										
Column1	pi_c	alpha	Tt4	Afterburner	eta_o	eta_p	eta_t	F	F_s	TSFC
1	24.25	1	2400	ON						
2	13.5	1	3600	OFF						
3	24.25	0	2400	OFF						
4	13.5	2	1200	OFF						
5	13.5	0	3600	ON						
6	24.25	2	2400	OFF						
7	13.5	2	3600	ON						
8	35	2	3600	OFF						
9	24.25	2	1200	ON						
10	35	1	1200	OFF						
11	13.5	0	1200	OFF						
12	13.5	1	1848	ON						
13	35	2	2400	ON						
14	35	0	1200	ON						
15	27.045	1	3600	ON						
16	35	0	3600	OFF						

The design chosen was a 16 run experiment with the four factors noted with variable levels. The outputs to be measured are noted in the columns to the left of the table.

The next step was to design the experiments determined by JMP into the DEVS-Suite Experimental Framework. By use of the "calibration feature" and the "value set features", the design was easily created. An example design is shown below in Figure 31. The "Outline" to the right of the design set-up confirms the factors, variables, variable levels, and the number of runs.

Next, the DOE design was run and outputs collected for analysis. The framework makes for an easy task of this. The data was output as before in both ".csv" and text formats and the results are next analyzed in JMP once again. The screening design takes the measured outputs and analyzes them for interaction effects. Much information is output in JMP, however, a sample can be seen below in Figure 32. The interaction effects show a correlation of several of the factor levels in combination with each other on the output of the turbojet engine (i.e. P-Values less than 0.06 are of significant interest).

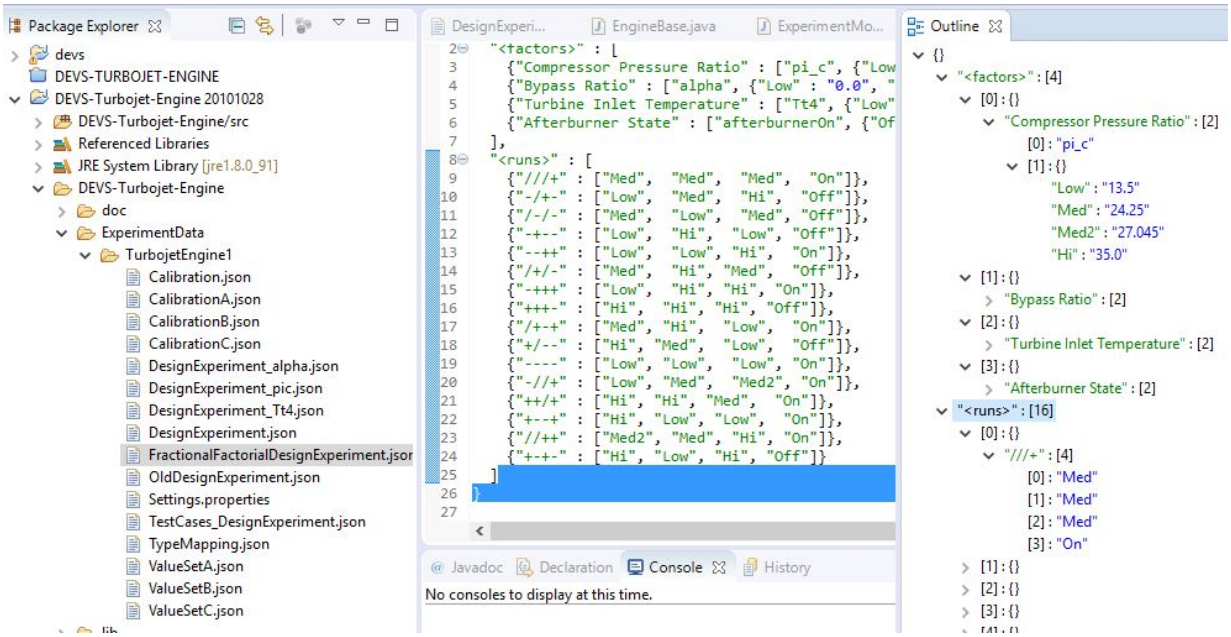


Figure 31: Design of Experiments — DEVS Turbojet Engine

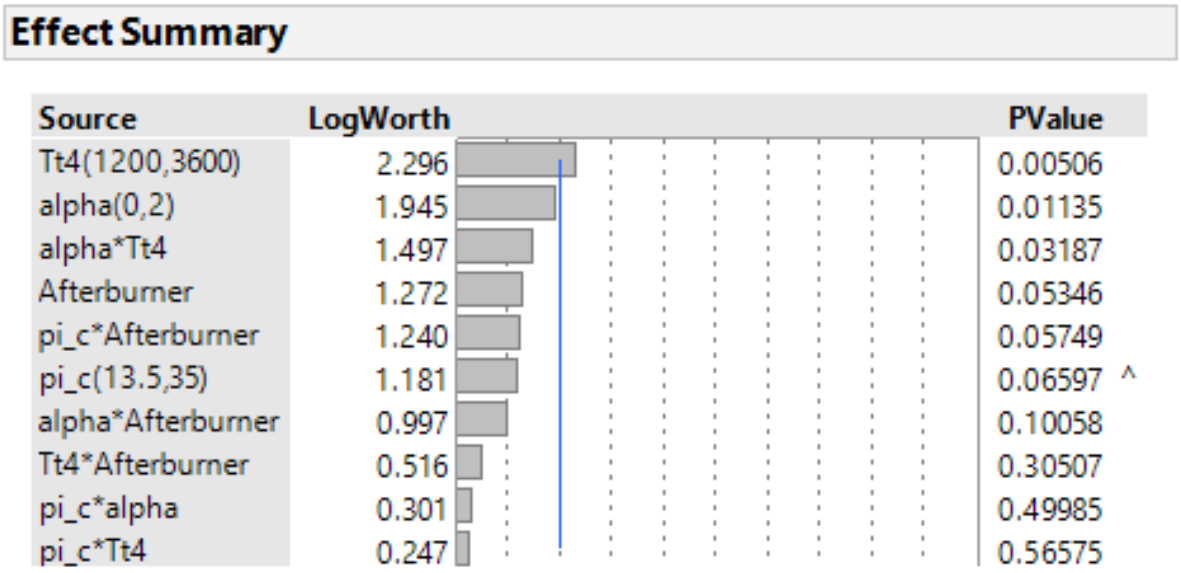


Figure 32: JMP Effect Screening Summary

Between the DEVS Experimental runs and the JMP statistical software, we can easily put together an experiment and analysis and begin to determine some interesting effects on the turbojet engine behavior. From this point, we can move to design more experiments and tailor them to the research of interest. This also allows us to move into Phase III of our M&S Design of Experiment strategy, as explained next.

To reiterate our goal for Phase III:

- We aim to optimize our turbojet engine for long duration, supersonic flight, with fuel savings and sufficient thrust output.

Based on findings from Phase II:

- Are there optimal levels of the variables which offer the greatest potential of performance enhancement?
- Creation of Randomized 'Optimal' Design of Experiments with continuous levels for factor variables

Again, we can use the help of JMP software to begin with a baseline of our optimization. We can specify our "Desirability" goals, or which variable outputs that we would like to maximize or minimize, or even reach a specific level somewhere in-between the maximum and minimum. In this case, we look to maximize the turbojet efficiencies and minimize the fuel spent. We would also like to maximize the thrust output. A sample of our specifications and a systematically offered set of design parameters in JMP can be seen in Figure 33. The optimization factor levels give us a baseline to begin with, with the percentage level of "Desirability" met, and the predicted outputs for the measured variables. We can easily take these factor levels and design additional experiments in our DEVS-Suite Experimental Framework for more analysis. Some examples of the additional experimental runs included factor behaviors over the span of operation

for the turbojet engine. We can notice the capabilities of the framework design for providing much research support. Several output results can be seen (Figure 34, Figure 35, Figure 36).

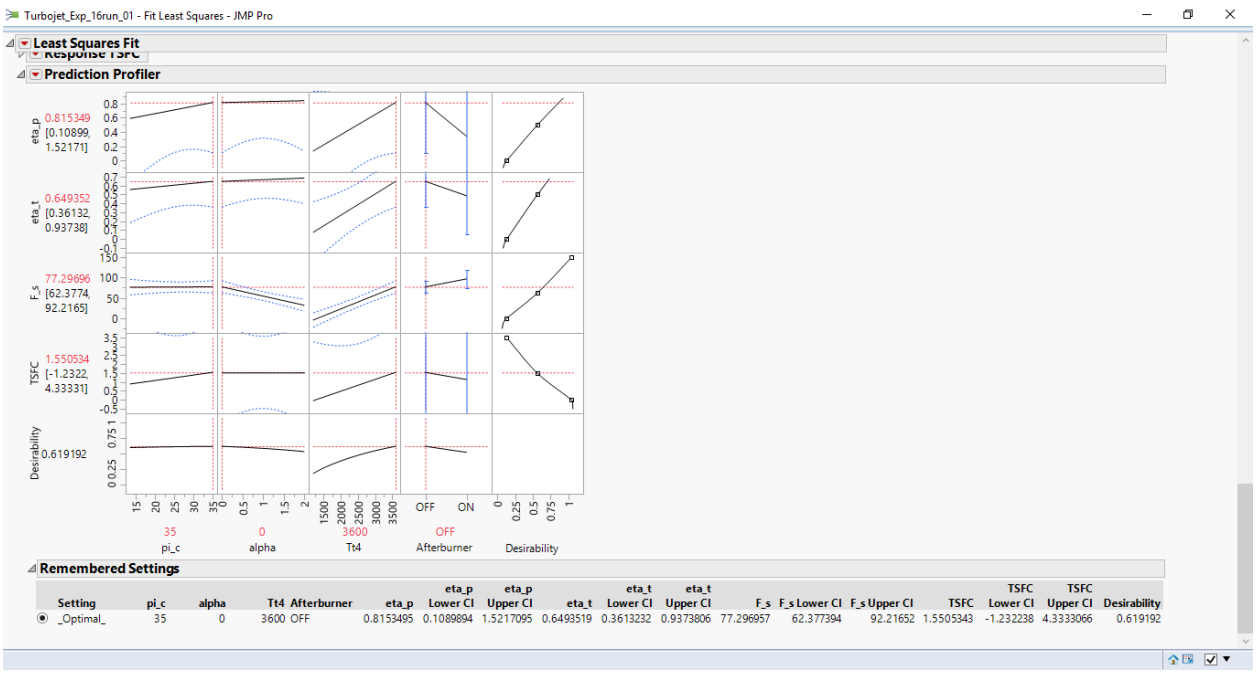


Figure 33: JMP Output — Optimizing Desirabilities

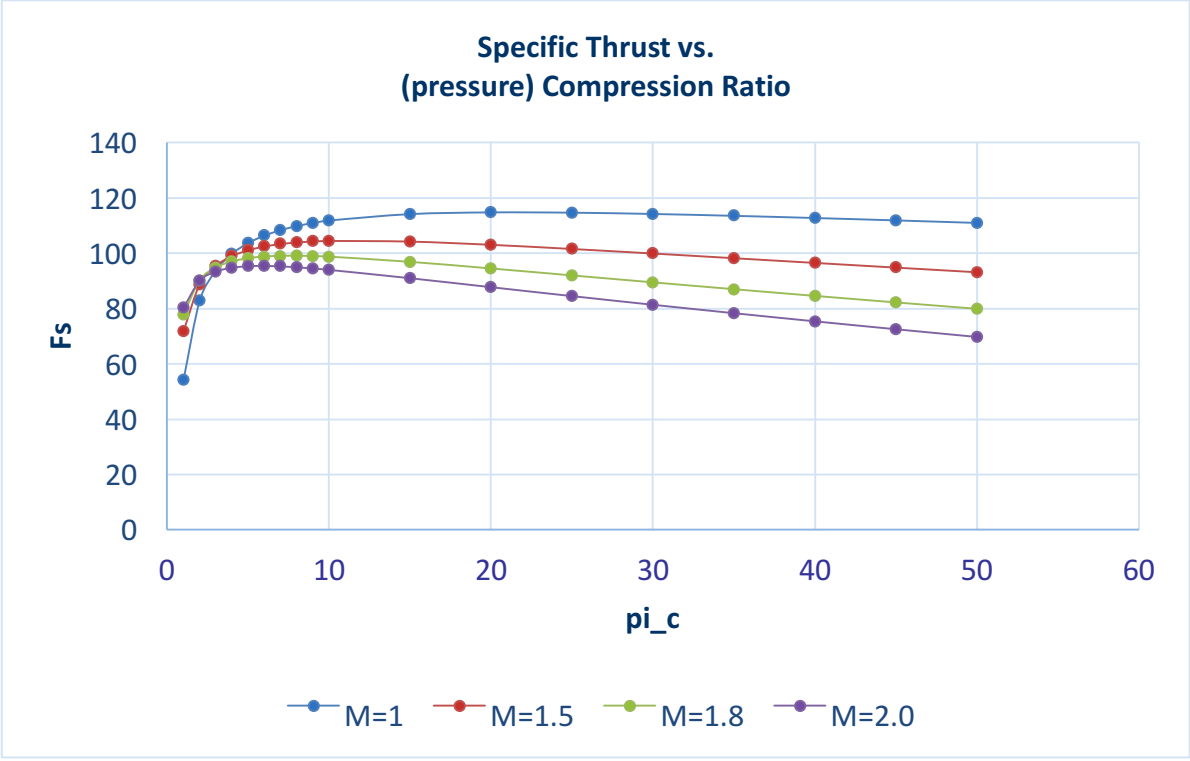


Figure 34: DEVS Output — Specific Thrust Factor Insight

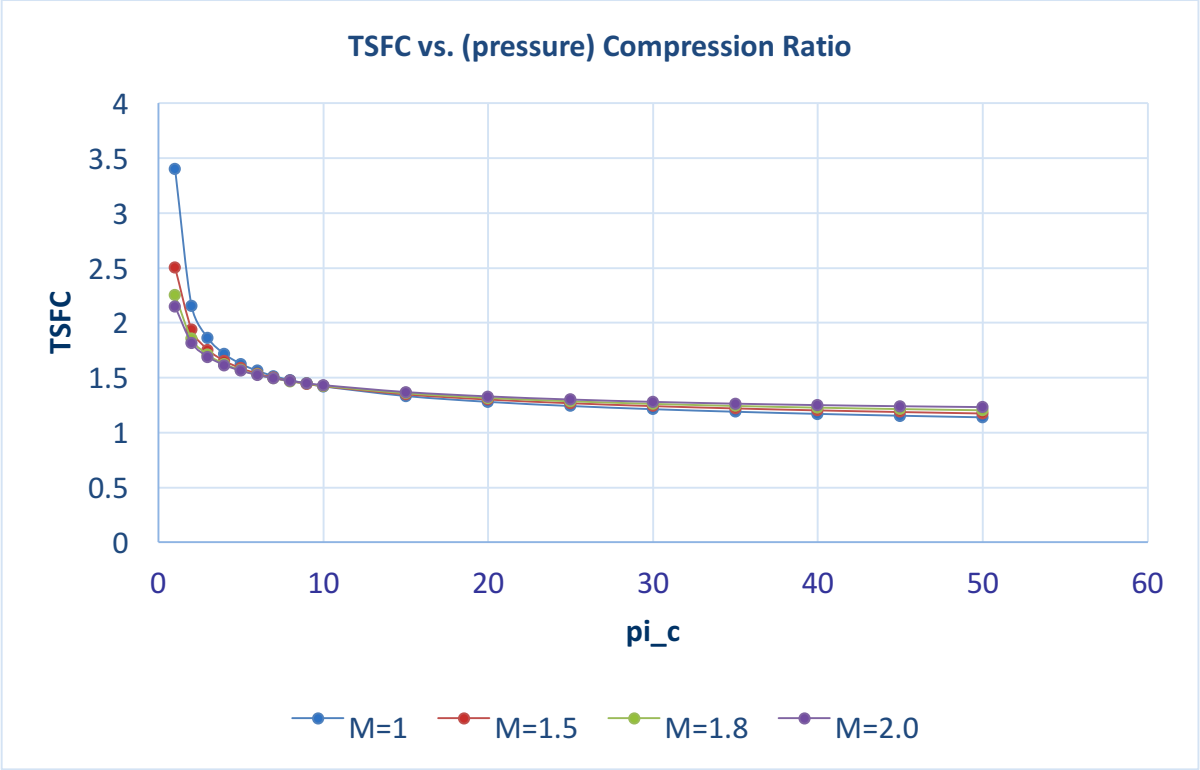


Figure 35: DEVS Output — TSFC vs. Compression Ratio Factor Insight

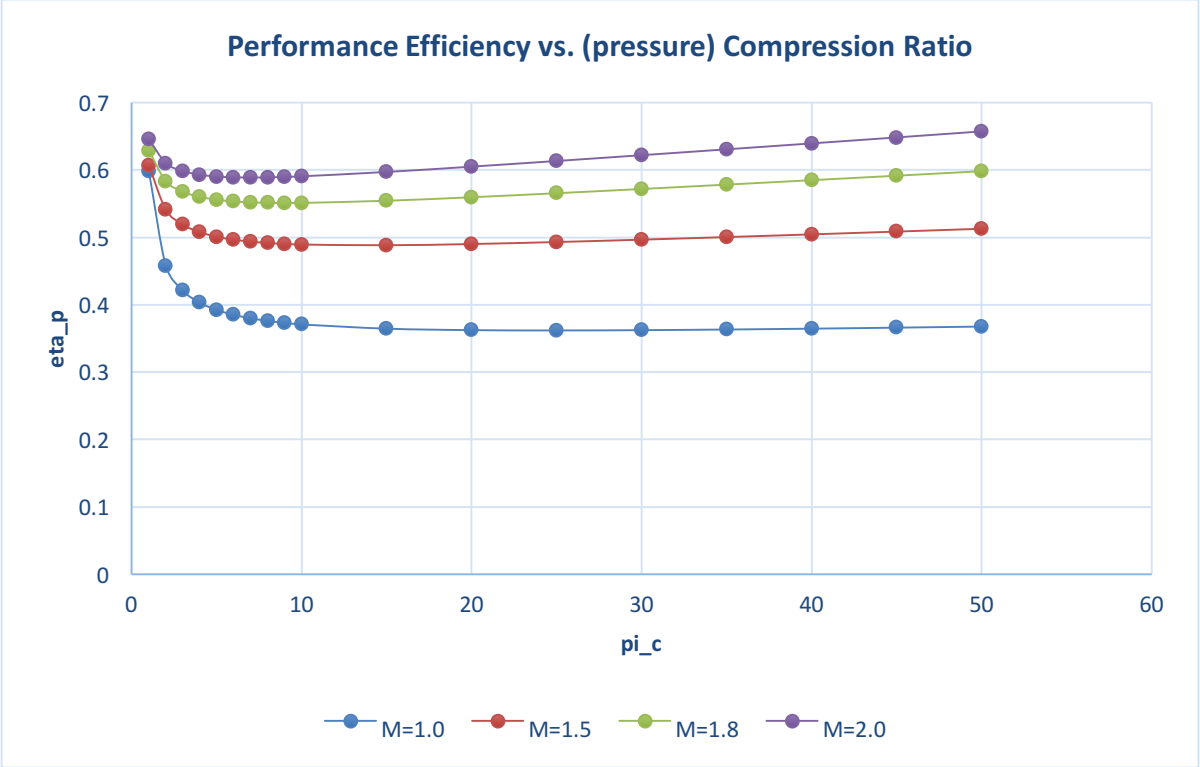


Figure 36: DEVS Output — Performance Efficiency vs. Compression Ratio Factor Insight

PROJECT MANAGEMENT

To complete this research and development endeavor, the project included as much project planning and Software Engineering "Best Practices" as possible for the short time frame. Some of the project planning activities included:

- Adopt A Lifecycle Model
- Identify The Tasks to Be Performed
- Analyze Dependencies
- Estimating:
  - Effort
  - Expertise
- "Assign" Responsibilities
- Map Tasks to a Timeline (Product Backlog and Sprints)
- Obtain Commitment from the Team on all decisions
- Facilitating Communication (Among the Team Members) - Skype
- Monitor and Report Status – Skype Meetings, email
- Assess Processes and Outcomes (e.g. Scrum Retrospects)

The project processes also informally followed the Project Management Process (taken from NASA Project Mgmt. Principals & Practices), and included as many good practices as possible. Some of the additional processes included were:

- Project Initiation
  - Define the Project
  - Develop a Project Vision
- Project Planning
  - Identify Skills and Commitments
  - Identify Reuse Opportunities
  - Develop Quality Plan
  - Identify Project Risks
- Project Execution
  - Manage and Track the Project
- Project Close-Out
  - Perform Project Close-Out

Since this was a distributed team effort, good communication was paramount to success. To allow the team to work more efficiently, a "Communications Plan" was created seen in Table 5.

Table 5: Communication Plan

Stakeholder Role/Name	Information Needs	How Frequent	Media	Person Responsible
Mentor/Acceptance Hessam Sarjoughian	Weekly status updates on schedule, progress, issues, questions	Every Friday at 12:00 AZ Time	Skype meeting + email	Team Members
Project Critique/ Research Team	Presentation on Project Description, Goals, Results, etc.	Scheduled presentation date	Skype Presentation	Team Members
Team Member/ Lucio Ortiz	Scrum standups, any immediate issues.	As necessary, (daily, every other day, weekly)	email	Blaze
Team Member/ Robert "Blaze" Blazewicz	Scrum standups, any immediate issues.	As necessary, (daily, every other day, weekly)	email	Lucio

Project Name

Modeling & Simulation of a Turbojet Engine
for Future Commercial Supersonic Transport
utilizing the DEVS-Suite Experimental Framework

Project Challenges

As with all projects, there were many challenges to complete this project on time, and satisfying the goals. Some of the challenges can be detailed as follows:

- Lofty vision
- Elaborate project plan for the Turbojet Engine
- The project plan was like a traditional startup business plan. A solution for a fictitious customer that quickly dies as soon as the product receives actual customer feedback.
- 10-weeks to develop project
- Defining the experiment engine took about 3-weeks leaving 6-weeks to reach a near final implementation.
    - Lot of pressure to develop a useful tool for both the Turbojet Engine and the general DEVS Suite.
    - Lots of uncertainty with the design path, and significant upfront chaos.

To tackle these challenges, the team had to take a somewhat unique approach to the design and development of our product. Categorizing the project as a cross-product between some of the more familiar development models, we can summarize our approach as follows:

- A Lean development process to develop a minimal viable product.
- A Rapid incremental prototyping based on customer feedback.
  - Customers: Lucio Ortiz and Professor Sarjoughian
- Use of Lean methodology
- Goal and milestones structured within 2-week Scrum sprints

With the help of some Agile project management tools, (i.e. Thoughtworks Mingle, and Microsoft Project), we organized and documented the development process, including goals, sprints, activities and tasks, milestones, and retrospectives/lessons learned. A snapshot view of the Mingle Agile project can be seen below in Figure 29. Collaborative tasks as well as individual user stories were defined, tracked, and documented in a sprint board with the team updating the product backlog as sprint iterations were completed.

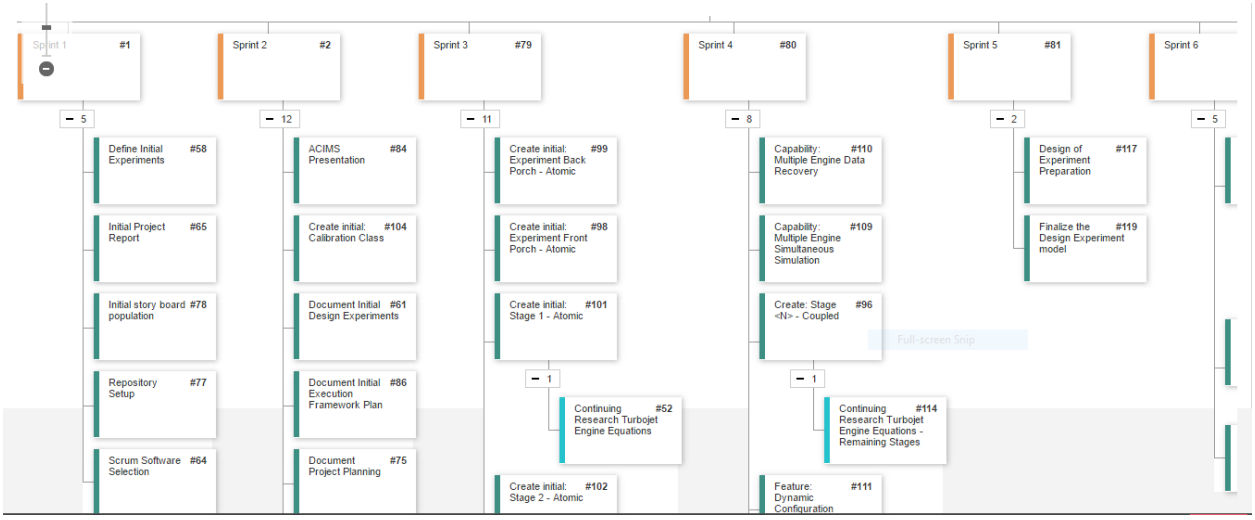


Figure 37: Mingle Agile Project — Sprint Board

Similarly, the team benefited from project retrospectives and lessons learned as the sprints moved along. A quick snap shot of some of these activities can also be seen in the Mingle Agile project below.

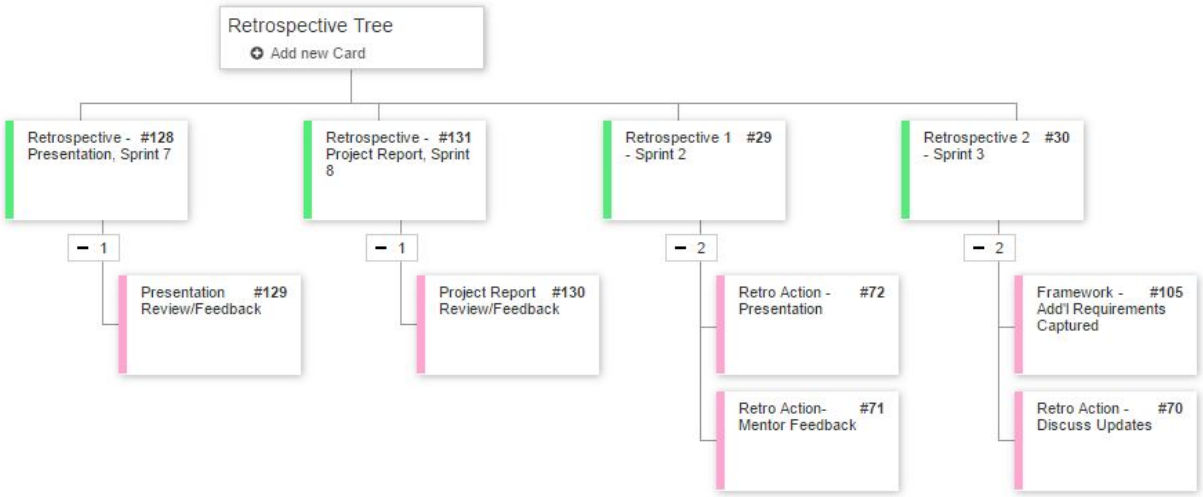


Figure 38: Mingle Agile Project — Retrospective Tree

Another useful tool to help manage the project timeline was the use of Microsoft Project. The Sprints were created with specific timelines and the major milestones were captured, as well as the major tasks and important milestones. A snapshot of the project timeline and Gantt chart can be seen below. These tools helped insure any product overruns or arising problems had sufficient resources to address them in a timely manner.

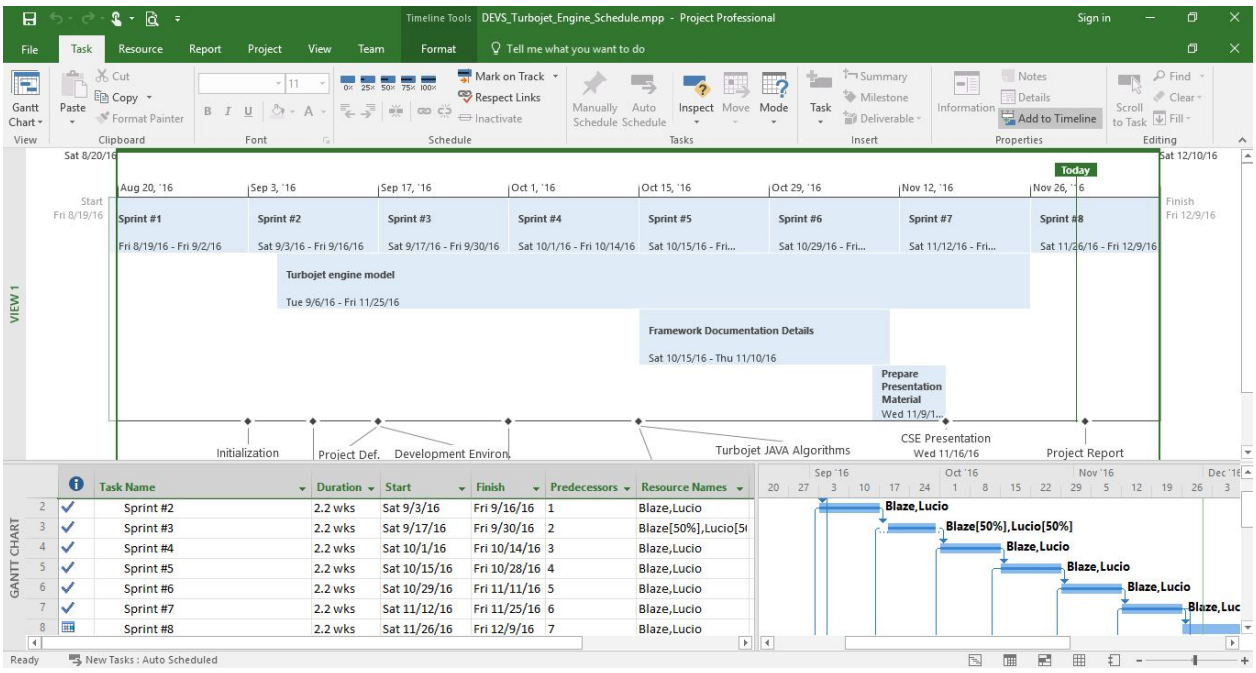


Figure 39: Microsoft Project — Turbojet Engine Schedule

CONCLUSIONS

The DEVS-Suite Experimental Framework proved to be a very capable tool for research on complex systems and system of systems. For the Turbojet Engine, the data and results obtained from the experiments utilizing the framework proved to give valuable insight to the factor interactions and the effects on engine behavior and operation. Some interesting results include the tradeoff studies between some performance variables vs. fuel efficiency and thrust. Insights into some future developments incorporating new materials for components, and even alternative fuels to be experimented with were also found. Future studies look to include full flight regimes with experimental runs from take-off to landing. We would also look to continue a model build, (also initiated during this project, but not implemented), of "Ramjet" and "Scramjet" turbojet operations. Also, ideas for other model types such as those concerning atmospheric conditions on other planets where new flight conditions will be imposed, are being conceptualized for possible research using the DEVS-Suite Experimental Framework.

Many software engineers are trained in the use of Design Experiments either through academia or through employer training for Six Sigma. In our experience, the training focuses on examples that are applicable to manufacturing. An anecdotal side effect is that software developers view Design Experiments as having little use within their domain. This is unfortunate because much insight and tuning of complex control systems would be gained through structured use of hypothesis-based experiments and an analysis of variance. We hope that the DEVS-Suite Experimental Framework will be further developed, formalized, and the effort eventually extended to support other modeling systems. This future progress would help facilitate the application of well-designed experiments and statistic evaluation to the software development domain.

REFERENCES

[1] S. Hargreaves, "Future of Travel, Supersonic jets can fly from N.Y. to L.A. in 2.5 hours (or less)," 26 November 2014. http://money.cnn.com/2014/11/26/luxury/supersonic-jet/

[2] Sarah Ramsey, "NASA Begins Work to Build a Quieter Supersonic Passenger Jet," 4 May 2016. http://www.nasa.gov/press-release/nasa-begins-work-to-build-a-quieter-supersonic-passenger-jet/

[3] G. J. Klir, "Architecture of Systems Problem Solving," New York, NY, Plenum Press, 1985.

[4] B. P. Zeigler, "Theory of Modeling and Simulation," New York, NY, Wiley Interscience, 1976.

[5] B. P. Zeigler, H. Praehofer and T. G. Kim, "Theory of Modeling and Simulation, 2nd Edition," San Diego, CA, Academic Press, 2000.

[6] J. D. Mattingly, W. H. Heiser, D. T. Pratt, "Aircraft Engine Design, 2nd Edition," Reston, VA, American Institute of Aeronautics and Astronautics, 2002.

[7] D. C. Montgomery, "Design and Analysis of Experiments, 8th Edition," Hoboken, NJ, John Wiley & Sons, 2013.

TECHNICAL ADVISORS

- Hessam Sarjoughian
  Associate Professor – Faculty
  School of Computing and Informatics
  Ira A. Fulton Schools of Engineering
  Arizona State University
- Soroosh Gholami
  Graduate Assistant/Associate
  School of Computing and Informatics
  Ira A. Fulton Schools of Engineering
  Arizona State University

A CAPTIONS

Table of Tables

Table of Figures

B DEVELOPMENT TOOLS

Modeling & Simulation Development Platform

DEVS-Suite Simulator, 2015 — 3.0.0
The Arizona Center for Integrative Modeling and Simulation
http://acims.asu.edu/software/devs-suite/

Software Language

Java SE Development Kit 8 — JavaSE 1.8
Java SE Runtime Environment 8 — JRE 1.8
Oracle Corporation : http://www.oracle.com/technetwork/java/javase/overview/

Integrated Development Platform

Eclipse Project — Neon 64bit
Eclipse Foundation : https://www.eclipse.org/eclipse/news/4.6/

Plugins used for coordination and improving software quality:

- Subversive
- Jautodoc
- FindBugs
- AnyEdit Tools
- Bracketeer
- Nodeclipse EditBox

UML Modeling

Astah Professional — 7.1
Change Vision : http://astah.net/editions/professional

Experiment Design and Analysis

JMP — 12.1
SAS Institute : http://www.jmp.com/en_us/software/data-analysis-software.html

Project Management

Mingle
Thoughtworks : https://www.thoughtworks.com/mingle/

Project Professional 2016
Microsoft : https://products.office.com/en-us/project/

Subversion hosting service
SourceRepo : http://sourcerepo.com/

C EXPERIMENTAL FRAMEWORK UML

The Experimental Framework is divided into two layers. The base layer with implementations for model's generator, acceptor, and transducer that interacts directly with the DEVS-Suite is the lower layer and is illustrated Figure 40. The upper layer is derived from the base layer and focuses on implementation of the framework's core functionality for the Front Porch (generator), Back Porch (accumulator), and Data Recorder (transducer). These are illustrated respectively in Figure 41, Figure 42, and Figure 43.

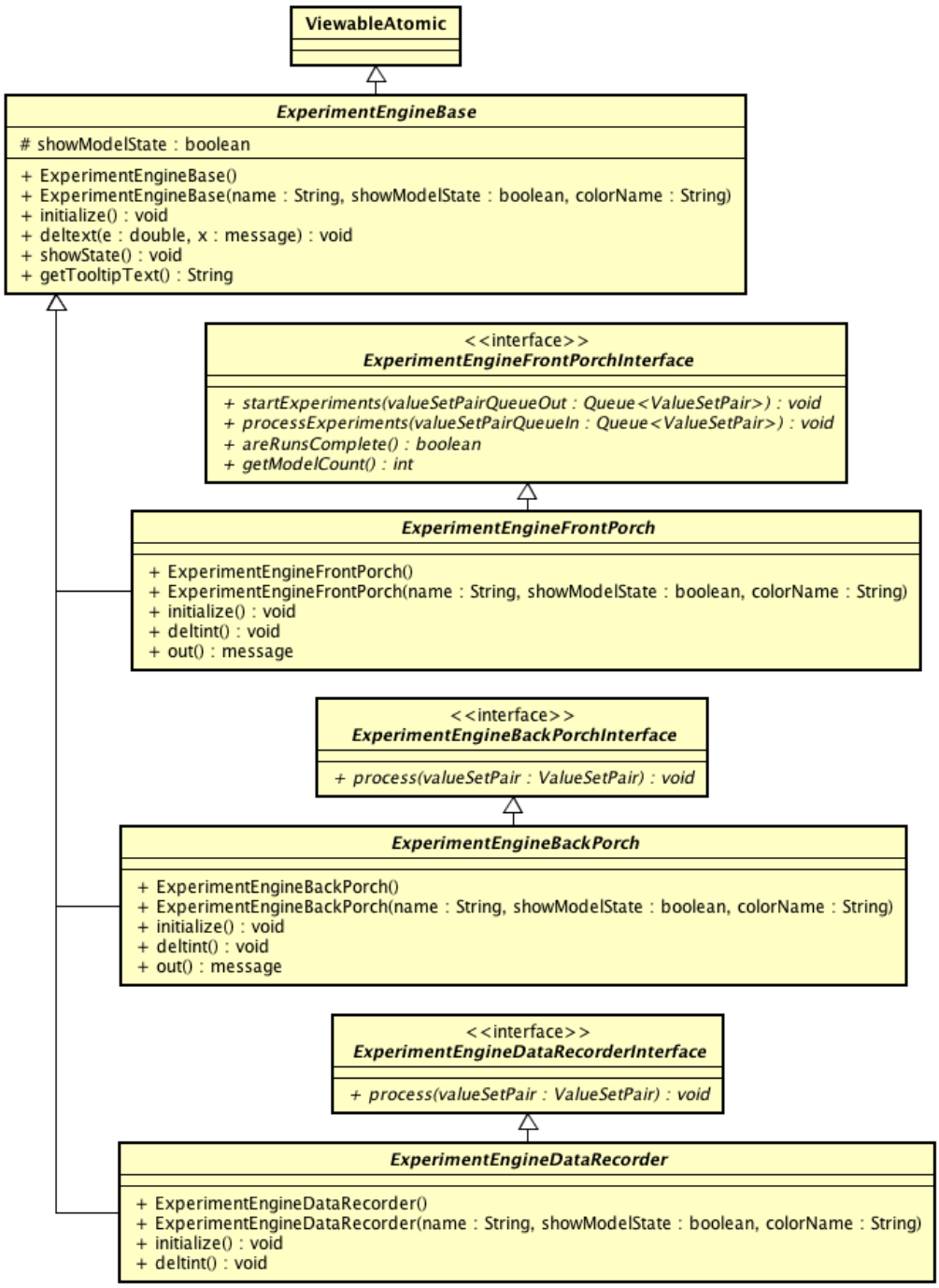


Figure 40: Experimental Framework base layer UML

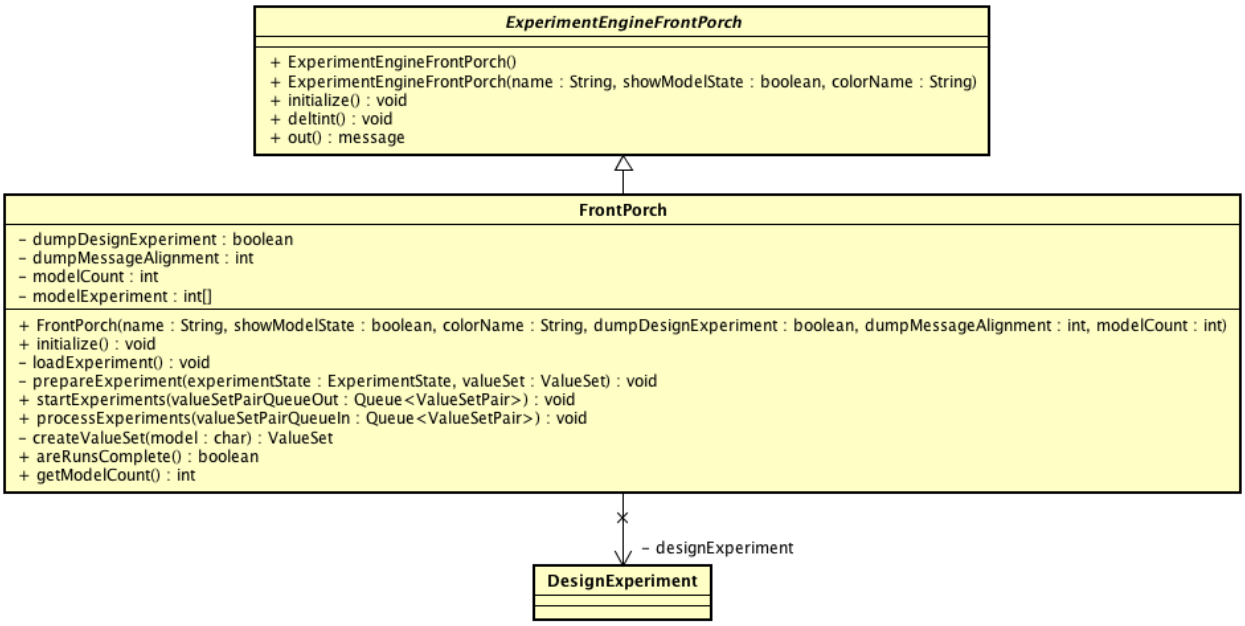


Figure 41: Front Porch class diagram

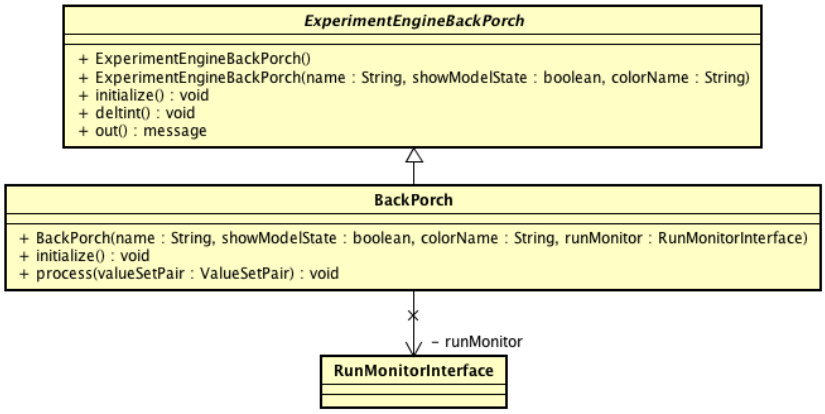


Figure 42: Back Porch class diagram

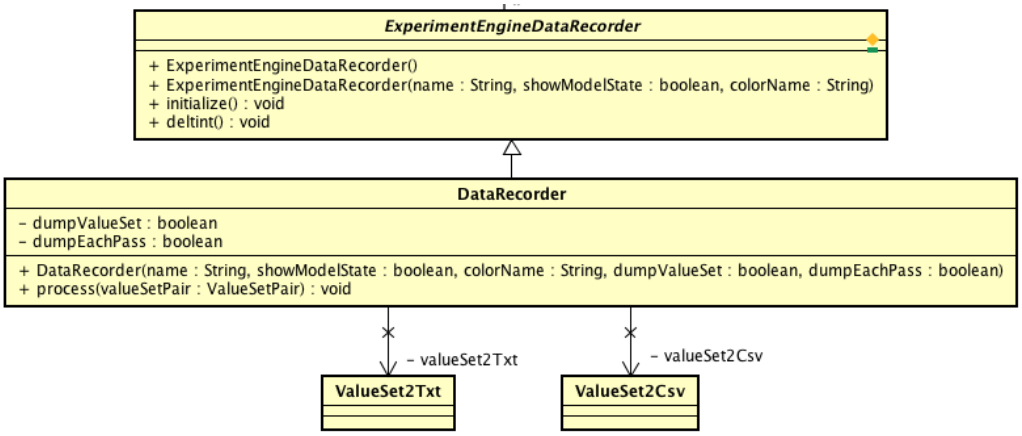


Figure 43: Data Recorder class diagram

D TURBOJET ENGINE NOZZLE EXAMPLE

The following code example demonstrates the value of using Java Reflection blended with DEVS-Suite to handle packing, transport, and unpacking of values within a model. The example demonstrates that the code for the Exhaust Nozzle stage of a Turbojet Engine with a polytropic/static supersonic fluid model is almost entirely focused on the math rather than the modeling infrastructure. There is one method invocation at the start and one at the end of the function that handle the writing/reading of the Java variables. The model developer's code is focused on the core computation, and announcement of values that are to be passed as output to the next model.

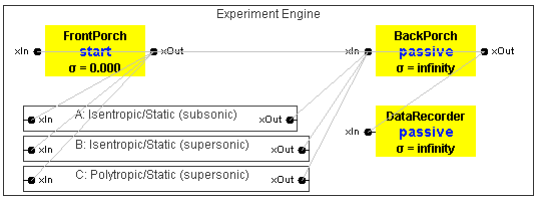
```
/**
 * Polytropic static supersonic.
 *
 * @param valueSet the value set
 */
public void PolytropicStaticSupersonic(final ValueSet valueSet) {
  valueSet.valuePop(this);

  // Expanded to Ambient Pressure

  // Pressure ratio at Nozzle Exit to Ambient
  Pzero_over_P7 = 0.9;

  // Pressure at Nozzle Exit
  P7 = Pzero/Pzero_over_P7;  //[lbf/ft^2]

  // Total Pressure at Nozzle Exit [lbf/ft^2]
  Pt7 = Pzero*pi_r*pi_d*pi_c*pi_b*pi_t*pi_n;

  // Total Temperature at Nozzle Exit [degR]
  T7 = Tzero* ((tao_lambda*tao_t) / (Math.pow((Pt7/P7),((gamma_t-1)/gamma_t))) * (Cp_c/Cp_t));

  // No loss in Static Pressure or Static Temperature (Isentropic)

  // Total Temperature at Nozzle Exit [degR]
  Tt7 = Tt6;

  // Temp. ratio at Nozzle
  tao_n = Tt7/Tt6;

  // Mach number at Nozzle Exit
  M_7 = Math.sqrt((2/(gamma_t-1)) * ((Math.pow((Pt7/P7),((gamma_t-1)/gamma_t)))-1));

  // Velocity ratio at Nozzle exit
  U_7_over_a_7 = M_7;

  // Velocity ratio over Ambient speed of sound
  U_7_over_a_zero = M_7*Math.sqrt((gamma_t/gamma_c) * (Rair_t/Rair_c) * (T7/Tzero));
  U_7_over_a_zero_squared = (U_7_over_a_zero*U_7_over_a_zero);

  if (afterburnerOn) {
    // fuel flow (ratio) with Afterburner ON
    f_ab = ((Cp_t*Tzero)/hpr) * (tao_lambda_ab-tao_r);

    // Temperature ratio Nozzle to Afterburner
    Tt7_over_Tt5 = (tao_lambda_ab/(tao_t*tao_lambda));

    // Velocity ratio at Nozzle exit to Ambient
    // ((U7_ab)/a0)^2
    U_7_ab_over_a_zero_squared = (Tt7_over_Tt5 * Math.pow(U_7_over_a_zero, 2));

    // (U7_ab)/a0
    U_7_ab_over_a_zero = Math.sqrt(U_7_ab_over_a_zero_squared);

    // U7_ab  Exhaust Velocity at Nozzle with AB ON [ft/s]
    U_7_ab = (Math.sqrt(U_7_ab_over_a_zero_squared))*azero;
```

```
  // U_0  Velocity at inlet (speed of aircraft) [ft/s]
  U_0 = azero*Mzero;

  // U_9 Velocity at Fan Nozzle Exhaust [ft/s]
  U_9 = (Math.sqrt(U_9_over_a_zero_squared))*azero;

  // Specific Thrust

  // Specific Thrust through core w/AB  [lbf/(lbm/s)]
  F_s_c_ab = (azero/gc) * ((1+f_ab) * (U_7_ab_over_a_zero)-Mzero+(1+f_ab) * (Rair_t/Rair_c) *
                           ((T7/Tzero) / (U_7_ab_over_a_zero)) * ((1-(Pzero/P7))/gamma_c));

  // (Total) Specific Thrust w/AB  [lbf/(lbm/s)]
  F_s_ab = ((1/(1+alpha))*F_s_c_ab  ) + ((alpha/(1+alpha))*F_s_fan);

  // (Total) Thrust [lbf]
  F_ab = F_s_ab*mdot_zero;

  // Thrust Specific Fuel Consumption (TSFC)  [1/hr]
  TSFC_ab = (1/(1+alpha)) * (f_ab/F_s_ab)*3600;

  // Thermal Efficiency of Turbojet Engine
  eta_t_ab = ((Math.pow((azero), 2)/gc) * ((1+f_ab) *  (Math.pow(U_7_ab_over_a_zero, 2)) +
             alpha*(U_9_over_a_zero_squared) - (1+alpha)*Math.pow(Mzero, 2))) / (2*f_ab*hpr*778.17);

  // Performance Efficiency of Turbojet Engine
  eta_p_ab = (2*Mzero)* (((1+f_ab) * (U_7_ab_over_a_zero) + (alpha) * (U_9_over_a_zero) - (1+alpha)*Mzero  ) /
             ((1+f_ab) * (U_7_ab_over_a_zero_squared) + (alpha) * (U_9_over_a_zero_squared) - (1+alpha) *
             (Mzero*Mzero)));

  // Overall Turbojet Engine Efficiency
  eta_o_ab = eta_t_ab*eta_p_ab;
} else {
  // Specific Thrust through core  [lbf/(lbm/s)]
  F_s_c = (azero/gc) * ((1+f) * (U_7_over_a_zero)-Mzero+(1+f) * (Rair_t/Rair_c) *
          ((T7/Tzero) / (U_7_over_a_zero)) * ((1-(Pzero/P7))/gamma_c));

  // Specific Thrust through fan  [lbf/(lbm/s)]
  F_s_fan = (azero/gc) * ((U_9_over_a_zero)-Mzero+(((T9/Tzero) / (U_9_over_a_zero)) *
            ((1-(Pzero/P9))/gamma_c)));

  // (Total) Specific Thrust  [lbf/(lbm/s)]
  F_s = ((1/(1+alpha))*F_s_c) + ((alpha/(1+alpha))*F_s_fan);

  // (Total) Thrust [lbf]
  F = mdot_zero*F_s;

  // Thrust Specific Fuel Consumption (TSFC) [1/hr]
  TSFC = (1/(1+alpha)) * (f/F_s)*3600;

  // Thermal Efficiency of Turbojet Engine
  eta_t = ((Math.pow((azero), 2)/gc) * ((1+f) * (Math.pow( U_7_over_a_zero,2  )) +
          alpha*(U_9_over_a_zero_squared) - (1+alpha)*Math.pow(Mzero, 2))) / (2*f*hpr*778.17);

  // Performance Efficiency of Turbojet Engine
  eta_p = (2*Mzero)* (((1+f) * (U_7_over_a_zero) + (alpha) * (U_9_over_a_zero) - (1+alpha)*Mzero  ) /
          ((1+f) * (U_7_over_a_zero_squared) + (alpha) * (U_9_over_a_zero_squared) - (1+alpha) *
          (Mzero*Mzero)));

  // Overall Turbojet Engine Efficiency
  eta_o = eta_t*eta_p;
}

if (valueSet.isFirstPass()) {
  valueSet.addValue(name, "Pzero_over_P7", Value.theType.eDouble, Pzero_over_P7, "[ft^2]",
                    "Pressure ratio at Nozzle Exit to Ambient", "P0/P7", true);
  valueSet.addValue(name, "P7", Value.theType.eDouble, P7, "[lbf/ft^2]", "Pressure at Nozzle Exit", "", true);
  valueSet.addValue(name, "T7", Value.theType.eDouble, T7, "[degR]", "Total Temperature at Nozzle Exit", "",
                    true);
  valueSet.addValue(name, "Pt7", Value.theType.eDouble, Pt7, "[lbf/ft^2]", "Total Pressure at Nozzle Exit", "",
                    true);
  valueSet.addValue(name, "Tt7", Value.theType.eDouble, Tt7, "[degR]", "Total Temperature at Nozzle Exit", "",
                    true);
  valueSet.addValue(name, "tao_n", Value.theType.eDouble, tao_n, "", "Temp. ratio at Nozzle", "", true);
  valueSet.addValue(name, "M_7", Value.theType.eDouble, M_7, "", "Mach number at Nozzle Exit", "", true);
```

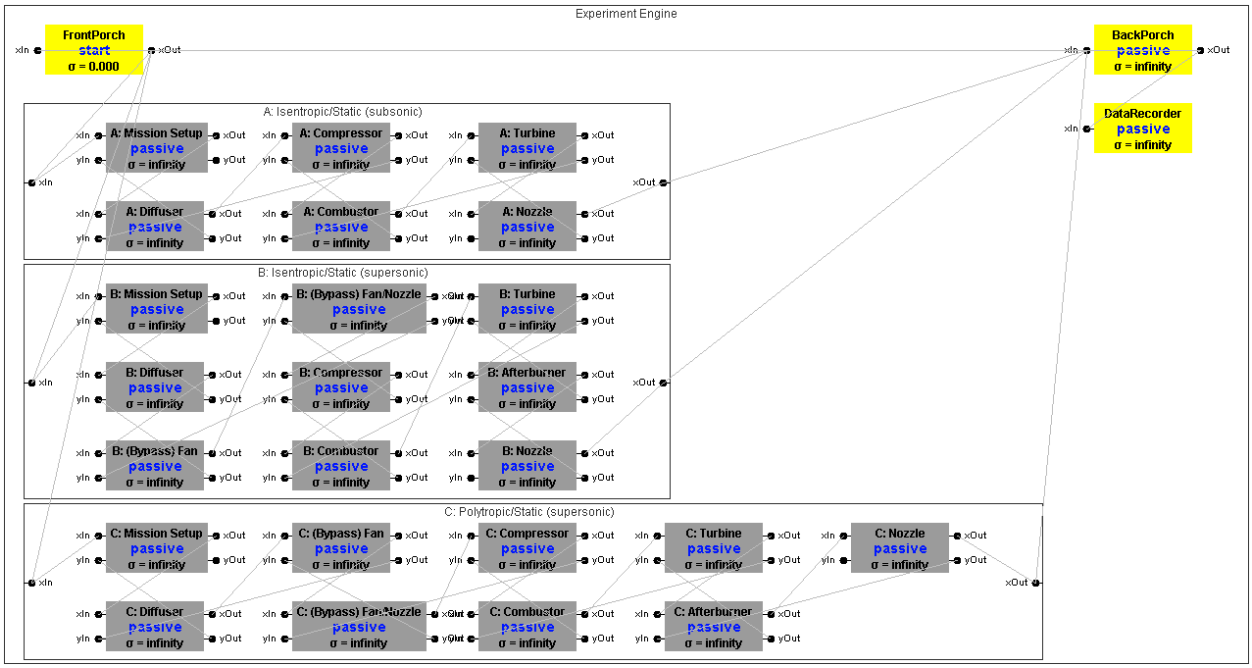
```
    valueSet.addValue(name, "U_7_over_a_7", Value.theType.eDouble, U_7_over_a_7, "",
                      "Velocity ratio at Nozzle exit", "U7/a7", true);
    valueSet.addValue(name, "U_7_over_a_zero", Value.theType.eDouble, U_7_over_a_zero, "",
                      "Velocity ratio over Ambient speed of sound", "U7/a0", true);
    valueSet.addValue(name, "U_7_over_a_zero_squared", Value.theType.eDouble, U_7_over_a_zero_squared, "",
                      "Velocity ratio at Nozzle exit to Ambient", "(U7/a0)^2", true);
    if (afterburnerOn) {
      valueSet.addValue(name, "f_ab", Value.theType.eDouble, f_ab, "", "fuel flow (ratio)", "", true);
      valueSet.addValue(name, "Tt7_over_Tt5", Value.theType.eDouble, Tt7_over_Tt5, "",
                        "Temperature ratio Nozzle to Afterburner", "Tt7/Tt5", true);
      valueSet.addValue(name, "U_7_ab_over_a_zero_squared", Value.theType.eDouble, U_7_ab_over_a_zero_squared,
                        "", "Velocity ratio at Nozzle exit to Ambient", "((U7_ab)/a0)^2", true);
      valueSet.addValue(name, "U_7_ab_over_a_zero", Value.theType.eDouble, U_7_ab_over_a_zero, "[ft/s]",
                        "Exhaust Velocity at Nozzle with AB ON", "(U7_ab)/a0", true);
      valueSet.addValue(name, "U_7_ab", Value.theType.eDouble, U_7_ab, "",
                        "Exhaust Velocity at Nozzle with AB ON", "", true);
      valueSet.addValue(name, "U_0", Value.theType.eDouble, U_0, "[ft/s]",
                        "Velocity at inlet (speed of aircraft)", "", true);
      valueSet.addValue(name, "U_9", Value.theType.eDouble, U_9, "[ft/s]", "Velocity at Fan Nozzle Exhaust", "",
                        true);
      valueSet.addValue(name, "F_s_c_ab", Value.theType.eDouble, F_s_c_ab, "[lbf/(lbm/s)]",
                        "Specific Thrust through core w/AB", "", true);
      valueSet.addValue(name, "F_s_ab", Value.theType.eDouble, F_s_ab, "[lbf/(lbm/s)]",
                        "(Total) Specific Thrust w/AB", "", true);
      valueSet.addValue(name, "F_ab", Value.theType.eDouble, F_ab, "[lbf]", "(Total) Thrust", "", true);
      valueSet.addValue(name, "TSFC_ab", Value.theType.eDouble, TSFC_ab, "[(lbm/hr)/lbf] or [1/hr]",
                        "Thrust Specific Fuel Consumption (TSFC)", "", true);
      valueSet.addValue(name, "eta_t_ab", Value.theType.eDouble, eta_t_ab, "",
                        "Thermal Efficiency of Turbojet Engine", "", true);
      valueSet.addValue(name, "eta_p_ab", Value.theType.eDouble, eta_p_ab, "",
                        "Performance Efficiency of Turbojet Engine", "", true);
      valueSet.addValue(name, "eta_o_ab", Value.theType.eDouble, eta_o_ab, "",
                        "Overall Turbojet Engine Efficiency", "", true);
    } else {
      valueSet.addValue(name, "F_s_c", Value.theType.eDouble, F_s_c, "[lbf/(lbm/s)]",
                        "Specific Thrust through core", "", true);
      valueSet.addValue(name, "F_s_fan", Value.theType.eDouble, F_s_fan, "[lbf/(lbm/s)]",
                        "Specific Thrust through fan", "", true);
      valueSet.addValue(name, "F_s", Value.theType.eDouble, F_s, "[lbf/(lbm/s)]", "Specific Thrust", "", true);
      valueSet.addValue(name, "F", Value.theType.eDouble, F, "[lbf]", "(Total) Thrust", "", true);
      valueSet.addValue(name, "TSFC", Value.theType.eDouble, TSFC, "[(lbm/hr)/lbf] or [1/hr]",
                        "Thrust Specific Fuel Consumption (TSFC)", "", true);
      valueSet.addValue(name, "eta_t", Value.theType.eDouble, eta_t, "", "Thermal Efficiency of Turbojet Engine",
                        "", true);
      valueSet.addValue(name, "eta_p", Value.theType.eDouble, eta_p, "",
                        "Performance Efficiency of Turbojet Engine", "", true);
      valueSet.addValue(name, "eta_o", Value.theType.eDouble, eta_o, "", "Overall Turbojet Engine Efficiency",
                        "", true);
    }
  }
  valueSet.valuePush(this);
}
```

E EXPERIMENTAL FRAMEWORK SCREENSHOTS

The following is an initial screenshot of a Turbojet Engine experiment with three gas models. Each of the coupled models under test are in blackbox mode to hide their details.



Here is an initial screenshot of the same Turbojet Engine experiment where the coupled models are displayed in detail.



Note: In the above screenshot, each of the coupled models has its own setting for the number of rows and from there the layout is dynamic. Also of note is that one of the coupled models has fewer engine stages than the other two.

F EXPERIMENT INPUT DATA SAMPLE

The following are input files of an example experiment for the DEVS-Suite Experimental Framework.

Input file: `DesignExperiment.json`

```
{
  "<factors>" : [
    {"Compressor Pressure Ratio" : ["pi_c", {"Low" : "12.2", "Hi" : "17.8"}]},
    {"Subsonic flight Mach" : ["Mzero", {"Low" : "0.7", "Hi" : "0.9"}]},
    {"Turbine Inlet Temperature" : ["Tt4", {"Low" : "1000", "Hi" : "2000"}]},
    {"Afterburner State" : ["afterburnerOn", {"Off" : "false", "On" : "true"}]}
  ],
  "<runs>" : [
    {"++++" : ["Hi",  "Hi",  "Hi",  "Off"]},
    {"+++-" : ["Hi",  "Hi",  "Hi",  "On"]},
    {"++-+" : ["Hi",  "Hi",  "Low", "Off"]},
    {"++--" : ["Hi",  "Hi",  "Low", "On"]},
    {"+-++" : ["Hi",  "Low", "Hi",  "Off"]},
    {"+-+-" : ["Hi",  "Low", "Hi",  "On"]},
    {"+--+" : ["Hi",  "Low", "Low", "Off"]},
    {"+---" : ["Hi",  "Low", "Low", "On"]},
    {"-+++" : ["Low", "Hi",  "Hi",  "Off"]},
    {"-++-" : ["Low", "Hi",  "Hi",  "On"]},
    {"-+-+" : ["Low", "Hi",  "Low", "Off"]},
    {"-+--" : ["Low", "Hi",  "Low", "On"]},
    {"--++" : ["Low", "Low", "Hi",  "Off"]},
    {"--+-" : ["Low", "Low", "Hi",  "On"]},
    {"---+" : ["Low", "Low", "Low", "Off"]},
    {"----" : ["Low", "Low", "Low", "On"]}
  ]
}
```

Input file: `Settings.properties`

```
# Settings.properties

# Number of concurrent experiment models to run. Range from 1 to 26.
ExperimentEngine_ModelCount=3

ExperimentEngine_BlackBoxModelWidth = 300
ExperimentEngine_BackgroundColor = yellow
ExperimentEngine_DumpDesignExperiment = false
ExperimentEngine_ShowModelState = false
ExperimentEngine_DumpValueSet = true
ExperimentEngine_DumpEachPass = false

#CalibrationPath=.\xxx\yyy
#CalibrationFile=MyCalibration.json
CalibrationDebug=false

#TypeMappingPath=.\aaa\bbb
#TypeMappingFile=MyTypeMapping.json

#ValueSetPath=.\aaa\bbb
#ValueSetFile=MyValueSet.json

ValueSet_Label=VSet
ValueSet_DumpMessageAlignment=60
```

Input file: `TypeMapping.json`

```
{
  "boolean" : ["Bool"],
  "char"    : [],
  "byte"    : [],
  "short"   : [],
  "int"     : ["Cnt","ms"],
  "long"    : [],
  "float"   : [],
  "double"  : ["double"],
  "String"  : ["Str"]
}
```

Input file: `CalibrationA.json`

```
{
  "KeTJET_Bool_BlackBox" : "false",
  "KeTJET_Bool_Afterburner" : "false",
  "KeTJET_Cnt_SimViewRows" : "2",
  "KeTJET_Str_FluidModel" : "Isentropic/Static (subsonic)"
}
```

Input file: `CalibrationB.json`

```
{
  "KeTJET_Bool_BlackBox" : "false",
  "KeTJET_Cnt_SimViewRows" : "3",
  "KeTJET_Bool_Afterburner" : "true",
  "KeTJET_Str_FluidModel" : "Isentropic/Static (supersonic)"
}
```

Input file: `CalibrationC.json`

```
{
  "KeTJET_Bool_BlackBox" : "false",
  "KeTJET_Cnt_SimViewRows" : "2",
  "KeTJET_Bool_Afterburner" : "true",
  "KeTJET_Str_FluidModel" : "Polytropic/Static (supersonic)"
}
```

Input file: `ValueSetA.json`

```
{
  "Isentropic/Static/Subsonic" : [
    { "Initial State" : [
      {"Tzero" : ["double", "389.97", "[degR]", "Inlet temperature", "", true]},
      {"Pzero" : ["double", "309.44", "[lbf/ft^2]", "Inlet pressure", "", true]},
      {"Mzero" : ["double", "2.0", "", "Supersonic flight Mach", "", true]},
      {"gamma" : ["double", "1.4", "[Cp/Cv]", "", "", true]},
      {"Rair" : ["double", "53.35", "[ft-lb/degR-lbm]", "Specific gas cnst. dry air", "", true]},
      {"gc" : ["double", "32.174", "[lbm-ft/s^2]", "gravity cnst", "", true]},
      {"Cp" : ["double", "0.24", "[Btu/lbm-degR]", "Specific heat of dry air", "", true]},
      {"pi_c" : ["double", "13.5", "[Pt3/Pt2]", "Compressor Pressure Ratio", "", true]},
      {"mdot_zero" : ["double", "170", "[lbm/s]", "Mass flow rate of air", "", true]},
      {"hpr" : ["double", "18400.0", "[Btu/lbm]", "Specified enthalpy of fuel", "", true]},
      {"Tt4" : ["double", "1669.670", "[degR]", "Turbine Inlet Temperature", "", true]}
    ]}
  ]
}
```

Input file: `ValueSetB.json`

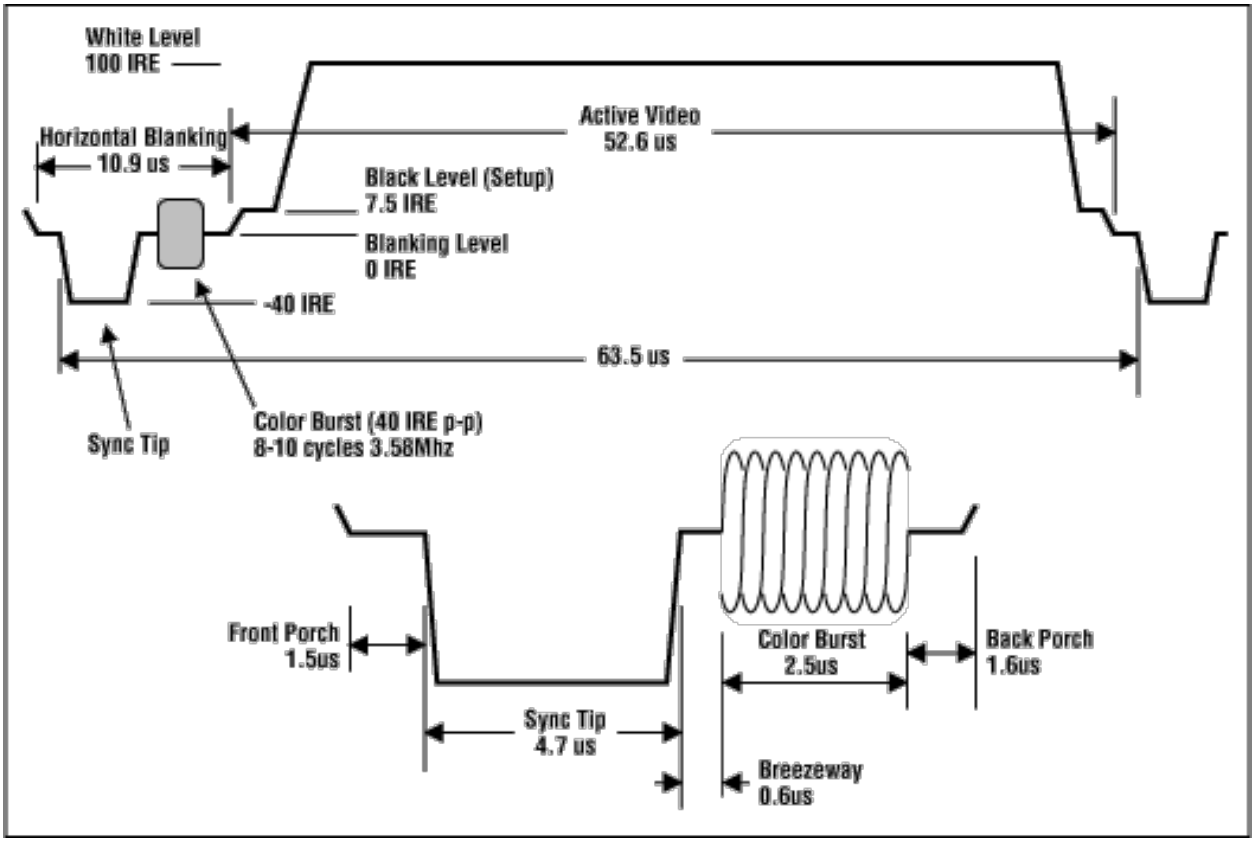
```
{
  "Isentropic/Static/Supersonic" : [
    { "Initial State" : [
      {"Altitude" :  ["double", "45", "[kft]", "", "", true]},
      {"Tzero" : ["double", "389.97", "[degR]", "Inlet temperature", "", true]},
      {"Pzero" : ["double", "309.44", "[lbf/ft^2]", "Inlet pressure", "", true]},
      {"Mzero" : ["double", "2.0", "", "Supersonic flight Mach", "", true]},
      {"gamma" : ["double", "1.4", "[Cp/Cv]", "(cool)", "", true]},
      {"Rair" : ["double", "53.35", "[ft-lb/degR-lbm]", "Specific gas cnst. dry air (cool)", "", true]},
      {"gc" : ["double", "32.174", "[lbm-ft/s^2]", "grav. cnst.", "", true]},
      {"Cp" :  ["double", "0.24", "[Btu/lbm-degR]", "Specific heat of dry air (cool)", "", true]},
      {"mdot_zero" : ["double", "170", "[lbm/s]", "Mass flow rate of air", "", true]},
      {"alpha" : ["double", "0.0", "[BPR]", "", "", true]},
      {"pi_fan" : ["double", "1.7", "", "", "", true]},
      {"pi_c" : ["double", "13.5", "[Pt3/Pt2]", "Compressor Pressure Ratio", "", true]},
      {"hpr" : ["double", "18400.0", "[Btu/lbm]", "Specified enthalpy of fuel", "", true]},
      {"Tt4" : ["double", "3600", "[degR]", "Turbine Inlet Temperature", "", true]},
      {"Tt6" : ["double", "4140.0", "[degR]", "Afterburner Inlet Temperature", "", true]},
      {"afterburnerOn" : ["boolean", "true", "", "Afterburner State", "", true]}
    ]}
  ]
}
```

Input file: `ValueSetC.json`

```
{
    "Polytropic/Static/Supersonic" : [
      { "Initial State" : [
        {"Altitude" :  ["double", "45", "[kft]", "", "", true]},
        {"Tzero" : ["double", "389.97", "[degR]", "Inlet temperature", "", true]},
        {"Pzero" : ["double", "309.44", "[lbf/ft^2]", "Inlet pressure", "", true]},
        {"Mzero" : ["double", "2.0", "", "Supersonic flight Mach", "", true]},
        {"Cp_c" :  ["double", "0.24", "[Btu/lbm-degR]", "Specific heat of dry air (cool)", "", true]},
        {"Cp_t" : ["double", "0.276", "[Btu/lbm-degR]", "Specific heat of dry air (heated)", "", true]},
        {"gamma_c" : ["double", "1.4", "[Cp/Cv]", "(cool)", "", true]},
        {"gamma_t" : ["double", "1.33", "[Cp/Cv]", "(heated)", "", true]},
        {"gc" : ["double", "32.174", "[lbm-ft/s^2]", "grav. cnst.", "", true]},
        {"hpr" : ["double", "18400.0", "[Btu/lbm]", "", "", true]},
        {"mdot_zero" : ["double", "170.0", "[lbm/s]", "Mass flow rate of air", "", true]},
        {"afterburnerOn" : ["boolean", "true", "", "Afterburner State", "", true]}
      ]},

      {"Polytropic Efficiencies" : [
        {"pi_d_max" : ["double", "0.99", "", "", "", true]},
        {"pi_b" : ["double", "0.96", "", "", "", true]},
        {"pi_n" : ["double", "0.99", "", "", "", true]},
        {"pi_fnz" : ["double", "0.99", "", "", "", true]}
      ]},

      {"Component Efficiencies" : [
        {"e_c" : ["double", "0.9", "", "", "", true]},
        {"e_f" : ["double", "0.89", "", "", "", true]},
        {"e_t" : ["double", "0.89", "", "", "", true]},
        {"eta_b" : ["double", "0.99", "", "", "", true]},
        {"eta_m" : ["double", "0.99", "", "", "", true]}
      ]},

      {"Afterburner & Fan Inputs" : [
        {"Tt4" : ["double", "3000.0", "", "", "", true]},
        {"pi_c" : ["double", "16.0", "", "", "", true]},
        {"pi_fan" : ["double", "2.9", "", "", "", true]},
        {"alpha" : ["double", "0.3", "[BPR]", "", "", true]},
        {"Tt6" : ["double", "4140.0", "", "", "", true]}
      ]}
    ]
}
```

G NOMENCLATURE

The nomenclature for the Experiment Framework's Front Porch and Back Porch is derived from terminology used by analog television signal encoding. Each horizontal line of encoded video has a region synchronization and a region of active video. The active video portion encoded two of the three red/green/blue colors and audio. The 3rd color was computed within the television using the principle of Pythagorean's theorem. This active portion of the signal was dynamic and contained a slice of the image. The synchronization portion, in contrast, is a highly structured and predictable region of the signal. Every horizontal line of the signal that encoded video repeated the structure of an active region bookended by a front porch and a back porch. We liken the Experiment Framework to the synchronization portion of the video signal which performs setup for a coupled model, and follows the coupled model as well structured bookends in contrast to the content of the coupled model which is dynamic like the active video. As the video signal is composed of a sequence of horizontal lines, so the Experiment Framework produces a repeating sequence of Front Porch (generator), to coupled model(s), to Back Porch (acceptor).



Note: An analog video receiver actually processes a horizontal line in the order of back porch, active video, and then the front porch, but it is common to think in terms of left-to-right as viewed in the diagram above and on an oscilloscope.

H BIOGRAPHIES

Lucio Ortiz

He works for the National Aeronautics and Space Administration (NASA) at the Neil A. Armstrong Flight Research Center in Edwards, California as a Test Information Engineer in the Mission Control, Range Engineering Branch working on experimental flight testing for future aircraft design. He holds a BSE in Aerospace Engineering/Aeronautics from Arizona State University and is pursuing a Master of Engineering in Modeling and Simulation also from Arizona State University. He has been an Aerospace Engineer for 6 years and has traveled between New Mexico and Arizona, working at both Sandia National Laboratories developing and testing neutron output for our nation's nuclear research, and with NASA providing research in aeronautics. When he is not busy with research, he frequently travels from California to Arizona to spend time with his two daughters. One attending Northern Arizona University and the other following closely behind, soon to graduate high school in Phoenix, Arizona. The motivation for this Master's Degree is to hopefully provide a good example to his daughters of what one can achieve with hard work and dedication.

Robert Blazewicz

He works for GM at the General Motors Proving Grounds in Milford Michigan as a Controls System Engineer working on Automated Driving & Active Safety Controls. He holds a BS in Computer Science from Northeastern University and is pursuing a Master of Engineering in Embedded Systems from Arizona State University. He has been a Software Engineer for 26 years and is an IEEE Certified Software Development Professional. Over his career he has worked in diverse areas of SIGINT, supercomputer databases & compilers, health care, and television data encoders. He is Secretary for the IEEE Southeast Michigan Section's chapter of Aerospace and Electronic Systems Society. As an adult leader in Boy Scouts, he is an Assistant District Commission of New Units and a Unit Commissioner. When not in Michigan, he lives in Tucson Arizona with his family. Married for 24 years, he has two daughters. One attending the University of Arizona and the other at the Northern Arizona University — he truly has a house divided.

This Page Intentionally Left Blank