# INTEGRATED ARCHITECTURE ANALYSIS AND TECHNOLOGY EVALUATION FOR SYSTEMS OF SYSTEMS MODELED AT THE SUBSYSTEM LEVEL

A Thesis
Presented to
The Academic Faculty

by

Douglas James Trent

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Aerospace Engineering

Georgia Institute of Technology
December 2017

# INTEGRATED ARCHITECTURE ANALYSIS AND TECHNOLOGY EVALUATION FOR SYSTEMS OF SYSTEMS MODELED AT THE SUBSYSTEM LEVEL

*To my family*

*For never letting me give up*

# ACKNOWLEDGEMENTS

I would like to take a moment to acknowledge and thank those who have contributed to me reaching this point in my life. This was not just six years of dedication and hard work on my part. It has been a culmination of thirty years of my life, with many people supporting me along the way.

I would first and foremost like to thank my academic advisor, Dr. Dimitri Mavris, who I am forever indebted to. You gave me the opportunity to prove I was up to the challenge of earning my Ph.D.. I would also like to thank the other members of my committee: Dr. Daniel Schrage, Dr. Alicia Sudol, Dr. Thomas Percy, and Mr. Mark Rogers. Your time and commitment to my research has helped shape this document into what it is.

To my colleagues at ASDL and NASA Marshall, thank you for providing me with an environment that was supportive through the entire process. Specifically, I would like to thank Mr. Stephen Edwards and Mr. Mark Rogers. Stephen, you helped to bring me into your division at ASDL, allowing me to work on tasks which were meaningful to me. You also provided me with technical guidance which was invaluable during my research. Mark, you brought me into your branch at NASA Marshall and provided an environment which allowed me to pursue this body of work while also supporting my professional growth and development. The two of you allowed for alignment of my academic and professional tasks and goals in a way most can only dream of.

On a more personal level, I would like to thank my family. Mom and Dad, without your years of guidance and support, I would never have made it here. You provided me with the absolute best opportunity to pursue my academic endeavors. You continue to support and push me to reach for my dreams, whatever they may be, the way only parents can. My brothers, Chris, Brian, and Greg, you always seem to have the right "encouragement" for the right time. Chris, you once wrote a blog post titled "Landing a man on the Moon and returning him safely to Earth", where you state *"It's my duty to be his highest hurdle, so that he will not be condemned to mediocrity. And no brother of mine is mediocre!"*. It is words such as these which echo in my mind and continue to be a driving force in all I do. I thank all of you for being the best family I could ask for.

I would also like to thank my friends. Nick Simone and Riley Driskel, you two have been some of my biggest outside supporters through this whole process. You lifted me up when I was down in the trenches of my everyday research and gave me encouragement to pick up and keep going, to achieve something greater than average. Nathan Knisely and Tyler Milner, your friendship and patience during my time at ASDL proved invaluable. You put up with me and helped me on a daily basis, during some of the most intense and challenging academic hurdles of my life, without any personal gain for yourselves. I can never repay that. Thank you.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# NOMENCLATURE

| | |
|---|---|
| **AADL** | Architecture Analysis and Design Language |
| **ACS** | Attitude Control System |
| **ADC** | Attitude Determination and Control |
| **ARCHITECT** | Architecture-Based Technology Evaluation and Capability Tradeoff |
| **ARCNET** | Architecture Resource-based Collaborative Network Evaluation Tool |
| **ATIES** | Abbreviated Technology Identification, Evaluation and Selection |
| **BLAST** | Beyond LEO Architecture Sizing Tool |
| **CCDev** | Commercial Crew Development |
| **C&DH** | Command and Data Handling |
| **CDR** | Critical Design Review |
| **CER** | Cost Estimating Relationship |
| **CONOP** | Concept of Operation |
| **COPA** | Computerized Orbital Performance Analysis |
| **COTS** | Commercial Orbital Transportation Services |
| **CPU** | Central Processing Unit |
| **DoCS** | Design of Computer Simulations |
| **DoD** | Department of Defense |
| **DoDAF** | Department of Defense Architecture Framework |
| **DOE** | Design of Experiments |
| **DYREQT** | Dynamic Rocket Equation Tool |
| **ECLSS** | Environmental Control and Life Support System |
| **ESA** | European Space Agency |
| **EVA** | Extravehicular Activity |
| **EXAMINE** | Exploration Architecture Model for In-Space and Earth-to-Orbit |
| **HExAM** | Human Exploration Architecture Model |

| | |
|---|---|
| **ICE** | Integrated Concurrent Engineering |
| **INCOSE** | International Council on Systems Engineering |
| **IntegrATE** | Integrated Architecture and Technology Exploration |
| **IPPD** | Integrated Product and Process Development |
| **IRMA** | Interactive Reconfigurable Matrix of Alternatives |
| **JCIDS** | Joint Capabilities Integration and Development System |
| **LDHEO** | Lunar Distant High Earth Orbit |
| **LDRO** | Lunar Distant Retrograde Orbit |
| **LEO** | Low Earth orbit |
| **LPF** | Layered Pareto Front |
| **MADM** | Multi-Attribute Decision Making |
| **MATE** | Multi-Attribute, Tradespace Exploration |
| **MBSE** | Model-Based Systems Engineering |
| **MDAO** | Multidisciplinary Design, Analysis, and Optimization |
| **MOA** | Matrix of Alternatives |
| **MPCV** | Multi-Purpose Crew Vehicle |
| **MT** | Metric Ton |
| **MYr** | Man-Years |
| **NAFCOM** | NASA/Air Force Cost Model |
| **NASA** | National Aeronautics and Space Administration |
| **NIST** | National Institute of Standards and Technology |
| **OMG** | Object Management Group |
| **OV** | Operational Viewpoint |
| **P-BEAT** | Process-Based Economic Analysis Tool |
| **PCEC** | Project Cost Estimating Capability |
| **PDR** | Preliminary Design Review |
| **PMF** | Propellant Mass Fraction |

| | |
|---|---|
| **QuantUM$^3$** | Quantitative Uncertainty Modeling, Management, and Mitigation |
| **RAAM** | Rapid Architecture Alternative Modeling |
| **ROSETTA** | Relational Oriented Systems Engineering and Technology Tradeoff Analysis |
| **RTG** | Radioisotope Thermoelectric Generator |
| **SAE** | Society of Automotive Engineering |
| **SEER** | Software for Evaluating and Estimating Resources |
| **SLS** | Space Launch System |
| **SoS** | System of Systems |
| **SRL** | System Readiness Level |
| **SRR** | System Requirements Review |
| **STASE** | Set Theory-Influenced Architecture Space Exploration |
| **STSD** | Set Theory-Influenced System Decomposition |
| **SV** | Systems Viewpoint |
| **SysML** | Systems Engineering Modeling Language |
| **TAPP** | Technology Alignment and Portfolio Prioritization |
| **TESSA** | Technique for the Enumeration of System of Systems Alternatives |
| **TIES** | Technology Identification, Evaluation and Selection |
| **TOPSIS** | Technique for Order Preferencing by Similarity to Ideal Solution |
| **TRIPS** | Technology Roadmapping and Investment Planning System |
| **TRL** | Technology Readiness Level |
| **TT&C** | Telemetry, Tracking, and Command |
| **UML** | Unified Modeling Language |
| **UTE** | Unified Tradeoff Environment |
| $\Delta m_i$ | Change in Inert Mass |
| $\Delta m_p$ | Change in Propellant Mass |
| $\Delta t$ | Change in Time |

| | |
|---|---|
| $\Delta \mathbf{V}$ | Change in Velocity |
| $\epsilon_{\mathbf{mli}}$ | Emissivity of Multi Layer Insulation |
| $\epsilon_{\mathbf{rad}}$ | Emissivity of Radiator Material |
| $\eta_{\mathbf{tr}}$ | Power Transmission Efficiency |
| $\overline{N}$ | Average Layer Density |
| $\rho_{\mathbf{rad}}$ | Radiator Aerial Density |
| $\mathbf{a}$ | System-Specific Constant |
| $\mathbf{A_{de}}$ | Design Envelope Area |
| $\mathbf{A_{mli}}$ | Surface Area of Multi Layer Insulation |
| $\mathbf{A_{rad}}$ | Radiator Area |
| $\mathbf{C_D}$ | Total Development Cost in Man-Years |
| $\mathbf{C_F}$ | Total Fabrication Cost in Man-Years |
| $\mathbf{C_R}$ | Material Radiation Correction Factor |
| $\mathbf{C_S}$ | Material Conductivity Correction Factor |
| $\mathbf{D}$ | Diameter |
| $\mathbf{d}$ | Number of Categories |
| $\mathbf{DF}$ | Degradation Factor |
| $\mathbf{F}$ | Element Fabrication Effort in Man-Years |
| $\mathbf{f_0}$ | Project System Engineering and Integration Factor |
| $\mathbf{f_1}$ | Technical Development Standard Correlation Factor |
| $\mathbf{f_2}$ | Technical Quality Correlation Factor |
| $\mathbf{f_3}$ | Team Experience Correlation Factor |
| $\mathbf{f_4}$ | Cost Reduction Factor Resulting from Learning Factor Application |
| $\mathbf{f_6}$ | Cost Growth Factor for Deviation from Optimum Time Schedule |
| $\mathbf{f_7}$ | Cost Growth Factor for Development by Parallel Contractors |
| $\mathbf{f_8}$ | Productivity Correction Factor |
| $\mathbf{f_k(X_k)}$ | Option Frequency of the kth Category of X |

| | |
|---|---|
| $F_s$ | Structure Factor |
| $G$ | Geometry |
| $g_0$ | Earth's Standard Gravity |
| $H$ | Element Development Effort in Man-Years |
| $I_{sp}$ | Specific Impulse |
| $L$ | Length |
| $M$ | Margin Factor |
| $m$ | Mass |
| $m_0$ | Initial Mass |
| $m_{IO}$ | Tank Inlet/Outlet Mass |
| $m_{active}$ | Active Thermal Control Hardware Mass |
| $m_{bac}$ | Broad Area Cooling Shield Mass |
| $m_{bare}$ | Bare Tank Mass |
| $m_{bat}$ | Power Storage Mass |
| $m_{bo}$ | Burnout Mass |
| $m_{cc}$ | Cryocooler Mass |
| $m_{circ}$ | Circulating Pump Mass |
| $m_{cool}$ | Engine Cooling Mass |
| $m_{core}$ | Nuclear Core Mass |
| $m_{ctrl}$ | Thermal Controller Mass |
| $m_{engines}$ | Mass of Engines/Thrusters |
| $m_{feed}$ | Propellant Feed Mass |
| $m_f$ | Final Mass |
| $m_{gauging}$ | Propellant Mass Gauging Device Mass |
| $m_{gen}$ | Power Generator Mass |
| $m_i$ | Inert Mass |
| $m_{lad}$ | Liquid Acquisition Device Mass |

| | |
|---|---|
| $\mathsf{m}_{\mathsf{misc}}$ | Mass of Miscellaneous Hardware |
| $\mathsf{m}_{\mathsf{mli}}$ | Multi Layer Insulation Mass |
| $\mathsf{m}_{\mathsf{nozz}}$ | Engine Nozzle Mass |
| $\mathsf{m}_{\mathsf{pass}}$ | Passive Thermal Control Hardware Mass |
| $\mathsf{m}_{\mathsf{pressurant}}$ | Pressurant Mass |
| $\mathsf{m}_{\mathsf{propmgt}}$ | Mass of Propellant Feed Management Hardware |
| $\mathsf{m}_{\mathsf{pwrmgt}}$ | Mass of Power Management Hardware |
| $\mathsf{m}_{\mathsf{p}}$ | Propellant Mass |
| $\mathsf{m}_{\mathsf{rad}}$ | Thermal Radiator Hardware Mass |
| $\mathsf{m}_{\mathsf{reg}}$ | Power Regulation and Distribution Mass |
| $\mathsf{m}_{\mathsf{sa}}$ | Tank Structural Attachments Mass |
| $\mathsf{m}_{\mathsf{sep}}$ | Tank Separation Mechanism Mass |
| $\mathsf{m}_{\mathsf{shield}}$ | Radiation Shield Mass |
| $\mathsf{m}_{\mathsf{tpa}}$ | Turbopump Assembly Mass |
| $\mathsf{m}_{\mathsf{trap}}$ | Propellant Trap Mass |
| $\mathsf{m}_{\mathsf{tubing}}$ | Tubing Mass |
| $\mathsf{m}_{\mathsf{vessel}}$ | Pressure Vessel Mass |
| $\mathsf{m}_{\mathsf{weld}}$ | Tank Weld Land Mass |
| $\dot{m}$ | Mass Flow Rate |
| $\mathsf{n}$ | Number of Units |
| $\mathsf{N}$ | Number of Design Points |
| $\mathsf{n}_{\mathsf{layers}}$ | Number of Multi Layer Insulation Layers |
| $\mathsf{n}_{\mathsf{tanks}}$ | Number of Propellant Tanks |
| $\mathsf{n}_{\mathsf{techs}}$ | Number of Technologies |
| $\mathsf{P}$ | Power |
| $\mathsf{p}$ | Propellant |
| $\mathsf{P}_{\mathsf{active}}$ | Active Thermal Control Power |

| | |
|---|---|
| $\mathbf{P_{cc}}$ | Cryocooler Power |
| $\mathbf{P_{circ}}$ | Circulating Pump Power |
| $\mathbf{P_{gauging}}$ | Propellant Mass Gauging Device Power |
| $\mathbf{P_{pass}}$ | Passive Thermal Control Power |
| $\mathbf{P_{req}}$ | Total Required Power Output |
| $\mathbf{p_{tank}}$ | Tank Pressure |
| $\mathbf{P_{t}}$ | Total Power |
| $\mathbf{Q}$ | Heat |
| $\mathbf{Q_{pwr}}$ | Power Subsystem Heat Load |
| $\mathbf{Q_{total}}$ | Total Heat Load |
| $\mathbf{r}$ | Radius |
| $\mathbf{r_{avg}}$ | Average Weighted Tank Radius |
| $\mathbf{R_{de}}$ | Design Envelope Radius |
| $\mathbf{S}$ | Similarity |
| $\mathbf{SF}$ | Safety Factor |
| $\mathbf{t}$ | Thickness |
| $\mathbf{T_{C}}$ | Cold Side Temperature |
| $\mathbf{T_{H}}$ | Hot Side Temperature |
| $\mathbf{t_{b}}$ | Burn Time |
| $\mathbf{U}$ | Ultimate Strength |
| $\mathbf{V}$ | Volume |
| $\mathbf{W}$ | Weight |
| $\mathbf{W_{bo}}$ | Burnout Weight |
| $\mathbf{w_{k}}$ | Weighting of the kth Category |
| $\mathbf{X}$ | System-Specific Cost-to-Mass Sensitivity Factor |
| $\mathbf{X_{k}}$ | Option of the kth Category of Design X |
| $\mathbf{Y_{k}}$ | Option of the kth Category of Design Y |

# SUMMARY

The first two decades of the twenty-first century have resulted in numerous redirections of United States space policy. This frequent redirection has produced challenges in the design and development of the systems of systems required for manned space exploration. Ever-changing design requirements leads to a lack of knowledge in the early phases of the design process. This lack of knowledge results in two primary challenges: overruns in cost and schedule due to frequent design changes and combinatorial explosion of alternatives due to large, discrete categorical design spaces. Due to the significant impact technologies have on the cost and schedule of a design, they should be considered during the conceptual design of systems of systems in an effort to reduce this lack of knowledge.

Current methods developed for the exploration of system of systems architectures and technologies define problems at the system level. However, in order to incorporate subsystem-level technology evaluation, architectures must also be defined at the subsystem level. Additionally, current methods developed for the purpose of technology evaluation do not support the exploration of large system of system design spaces. Therefore, a gap exists in current methods and frameworks to perform integrated architecture analysis and technology evaluation defined at the subsystem level.

To integrate architecture analysis and technology evaluation at the subsystem level, several questions and hypotheses were posed during a discussion of a general concept exploration process to guide the development of a new framework. However, in order to test these hypotheses, a digital test bed capable of performing integrated architecture analysis and technology evaluation at the subsystem level had to be selected. No tools were identified within the space transportation community which met this requirement. As a result, the Dynamic Rocket Equation Tool (DYREQT) and a collection of subsystem-level in-space transportation models were developed to provide a modeling and simulation environment capable of producing the necessary data for experimentation. DYREQT provides the capability to integrate user-developed subsystem models in a tool developed for space transportation architecture analysis and design.

Results from the experiments designed in response to the research questions and hypotheses led to conclusions which guided the definition of the Integrated Architecture and Technology Exploration (IntegrATE) framework. This new framework fulfills the research objective by providing integrated architecture analysis and technology evaluation at the subsystem level in an effort to increase design knowledge during the conceptual design process. IntegrATE provides flexibility such that it can be tailored to a wide range of problems. It also provides a high degree of transparency throughout to help reduce the likelihood of bias towards individual architectures or technologies. Finally, the IntegrATE framework and DYREQT were demonstrated on a notional manned Mars 2033 design study to highlight the utility of these new developments.

# CHAPTER I

# MOTIVATION

*"With our present knowledge, we can respond to the challenge of stellar space flight solely with intellectual concepts and purely hypothetical analysis. Hardware solutions are still entirely beyond our reach and far, far away."*

— Dr. Wernher von Braun[1]

Since before man became a spacefaring race, dreams of reaching deep into the vast unknown filled our imaginations. As technology has progressed to make these dreams an ever-closer reality, the machines designed and built to carry out such a task grow larger and more complex. As these designs grow ever greater in complexity, so does the number of potential designs that may fulfill a given task. The research detailed within this manuscript focuses on the conceptual phase of design. It is well known that decisions made early in the design process have disproportionate impacts on the future cost and schedule of corresponding programs. It is important to understand the interaction of the mission, the vehicle, and technology on a given architecture to ensure that all measures are taken such that a suitable architecture is selected to prevent costly financial and schedule overruns.

The motivations and background come from two main sources, human space exploration and space transportation architectures. The design of such architectures presents unique challenges not seen in other related fields and provides the basis for the research efforts that follow. However, this does not imply that the research herein is limited to only these motivating sources.

[1]Popular Science, Volume 183, July 1963 (p. 170)

## *1.1 U.S. Space Exploration Policy in the 21st Century*

The dawn of the 21st century has brought drastic changes to space policy. For decades, deep space exploration was relegated to robotic spacecraft, with manned missions confined to low earth orbit (LEO). However, the landscape began to change as policy makers became restless with the status quo and desired a shift to a more ambitious presence beyond Earth. At the start of the century, only two vehicles were qualified to fly humans into space, the United States-built Space Shuttle and the Russian-built Soyuz. The only clear destination in LEO was the International Space Station.

### 1.1.1 Presidential Remarks on U.S. Space Policy

In 2004, President George W. Bush ordered the retirement of the Space Shuttle fleet to make way for the development of a new launch vehicle [13]. This order marked the first major shift in United States space exploration policy in the 21st century. In his address, President Bush called for a new focus, stating three primary objectives:

1. Completion of the International Space Station
2. Development of a new manned exploration vehicle
3. Return to the moon as a launching point for missions beyond

Following these objectives, The International Space Station, in development since 1998, was scheduled to have its last major United States component installed by 2010, coinciding with the last flight of the Space Shuttle Program [76]. NASA also began work on the Constellation program to replace the aging Space Shuttle fleet. To support the transition process, NASA implemented the Commercial Crew Development program to stimulate private development of vehicles capable of launching humans into low earth orbit. However, budgetary and schedule overruns with the Constellation program led to a review of manned space flight, commissioned in 2009, to reassess the plans set forth by President Bush in 2004 [8].

### 1.1.2 Review of U.S. Human Space Flight Plans Committee

The 2009 Review of U.S. Human Space Flight Plans Committee, commonly referred to as the Augustine Commission, was officially tasked with developing suitable options for consideration by NASA regarding a human space flight architecture that would [8]:

1. Expedite a new U.S. capability to support utilization of the International Space Station
2. Support missions to the Moon and other destinations beyond LEO
3. Stimulate commercial space flight capability
4. Fit within the current budget profile for NASA exploration activities

Though the review focused heavily on programs and current launch vehicle hardware, it did little for giving clear direction. The report outlines three exploration paths:

1. Mars First, with a Mars landing, perhaps after a brief test of equipment and procedures on the Moon.
2. Moon First, with lunar surface exploration focused on developing the capability to explore Mars.
3. A Flexible Path to inner solar system locations, such as lunar orbit, Lagrange points, near-Earth objects and the moons of Mars, followed by exploration of the lunar surface and/or Martian surface.

The review outlined many findings, such as a need for program stability, mission and funding alignment, and commercial involvement. These findings would become the basis for a redirection of human exploration efforts moving forward.

### 1.1.3 NASA Authorization Acts of 2010 & 2017

The NASA Authorization Act of 2010 put into law many of the findings of the Augustine Commission. One of the first things it did was to cancel the Constellation Program which had been plagued by cost and schedule overruns resulting from years of budgetary cuts. Figure 1 shows this phenomenon of budget reductions in early years leading to budget overruns in later years to maintain a fixed schedule. However, not all elements would be scrapped. From the Constellation program, the Orion crew vehicle would be redesigned as the Multi-Purpose Crew Vehicle (MPCV). The act also ordered the development of the Space Launch System (SLS) to replace the Ares launch vehicles as the nation's vehicle for access to space [1]. The new vehicle would utilize commonality from both Space Shuttle and Ares heritage hardware and designs [95].

The development of this new system would still leave the U.S. with a significant time gap in domestic human space flight access, about seven years [8]. To reduces this gap, NASA was directed to continue investments in commercial entities to develop independent access to low earth orbit. For this, NASA continued the Commercial Orbital Transportation Services (COTS) program and the Commercial Crew Development Program (CCDev). These programs awarded contracts in phases to companies who developed and provided the requested contract services to NASA. U.S. companies such as Orbital Sciences, Space Exploration Technologies (SpaceX), Sierra Nevada Corporation, Blue Origin, and The Boeing Company were selected to demonstrate launch capabilities to NASA [80, 84].

The NASA Authorization Act of 2010 also redirected the focus of mission destinations. Where the Vision for Space Exploration gave a clear direction of lunar exploration leading to the surface exploration of Mars, the new policy directed NASA to follow the Flexible Path option from the Augustine Commission. This option was the most complex and least defined of the options presented. Figure 2 depicts the various

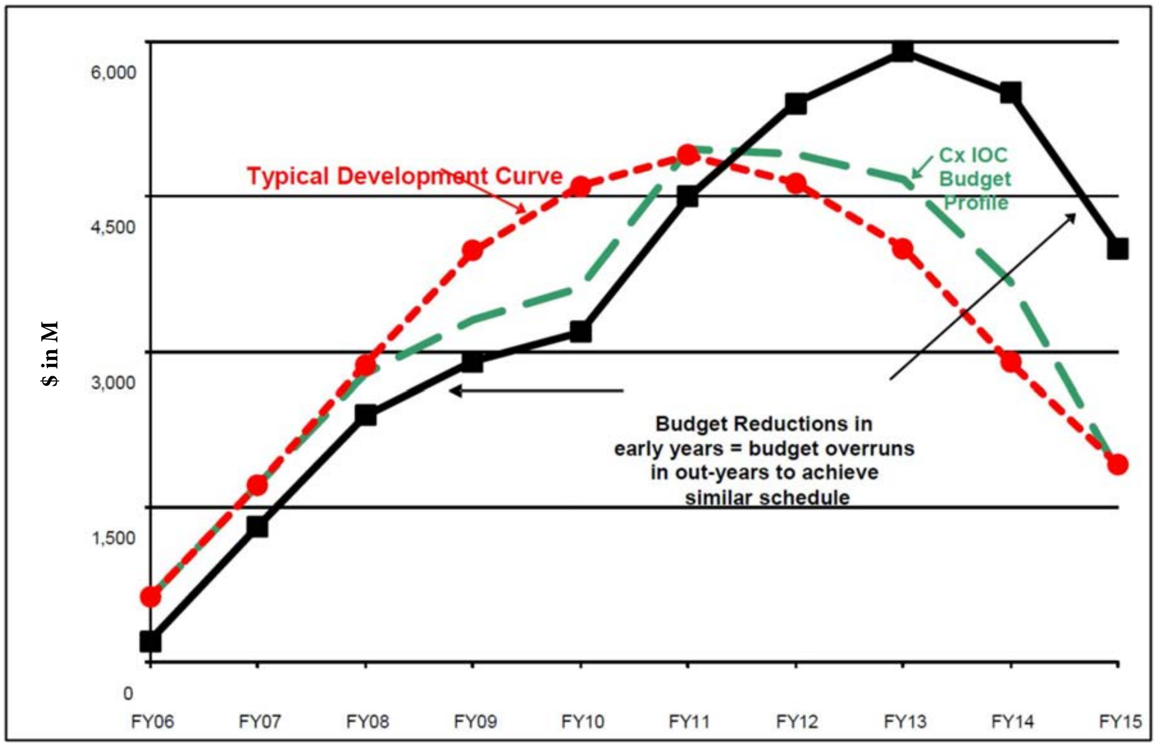


**Figure 1:** Constellation budget cuts prior to preliminary design review (PDR) resulted in a budget profile far less than optimal for a development project. The initial decrease in FY2010 was primarily due to limitations in funding due to Space Shuttle retirement efforts. The result was a drastic increase in later funding to maintain a fixed design schedule which ultimately fell behind and became a large factor in the cancellation of the Constellation program [95].

paths available to reach the final Mars surface destination. Although the concept was intended to provide, as its name implies, flexibility in destinations to reach the ultimate goal of Mars surface exploration, the lack of a clear direction would lead to further redirections. The Augustine Commission stated a need for program stability to maintain a return on investment [8].

Since the Authorization Act of 2010, the new launch capability, SLS, has had an unsteady funding profile, as seen in Figure 3. Political tension between the President and Congress has caused financial instability for the SLS program. Early funding cuts have led to cost overruns, as explained in Figure 1, as well as schedule slips. The first flight of SLS was originally scheduled for 2016, but was subsequently rescheduled for

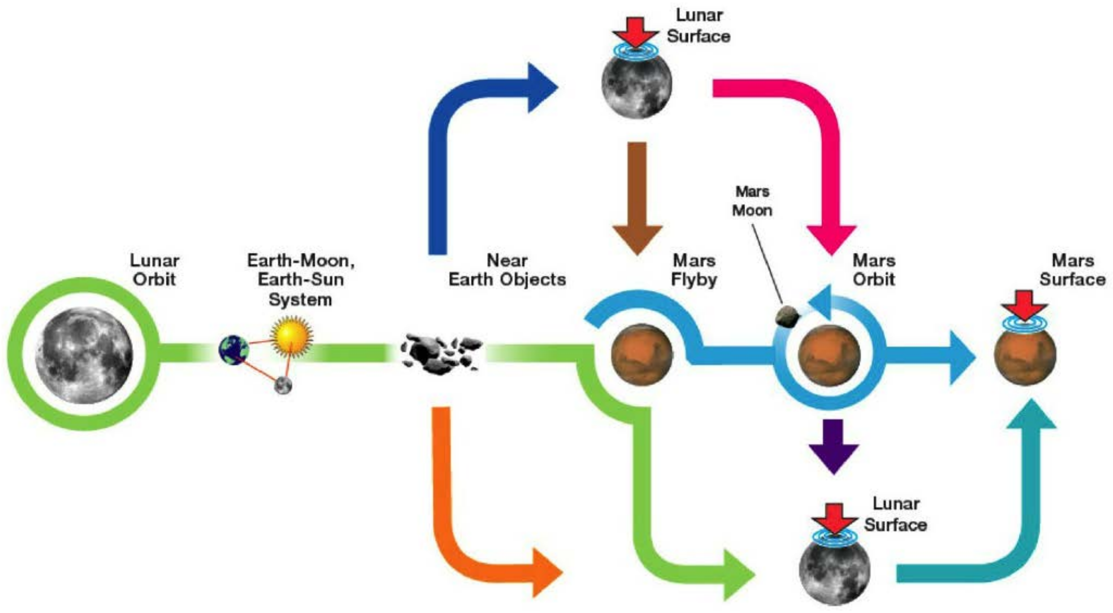


**Figure 2:** The Flexible Path option was proposed by The Review of U.S. Human Spaceflight Plans Committee as a viable exploration strategy for eventual manned Mars surface missions [8]. However, the lack of a clear and defined set of missions can lead to many future redirections down the different mission paths.

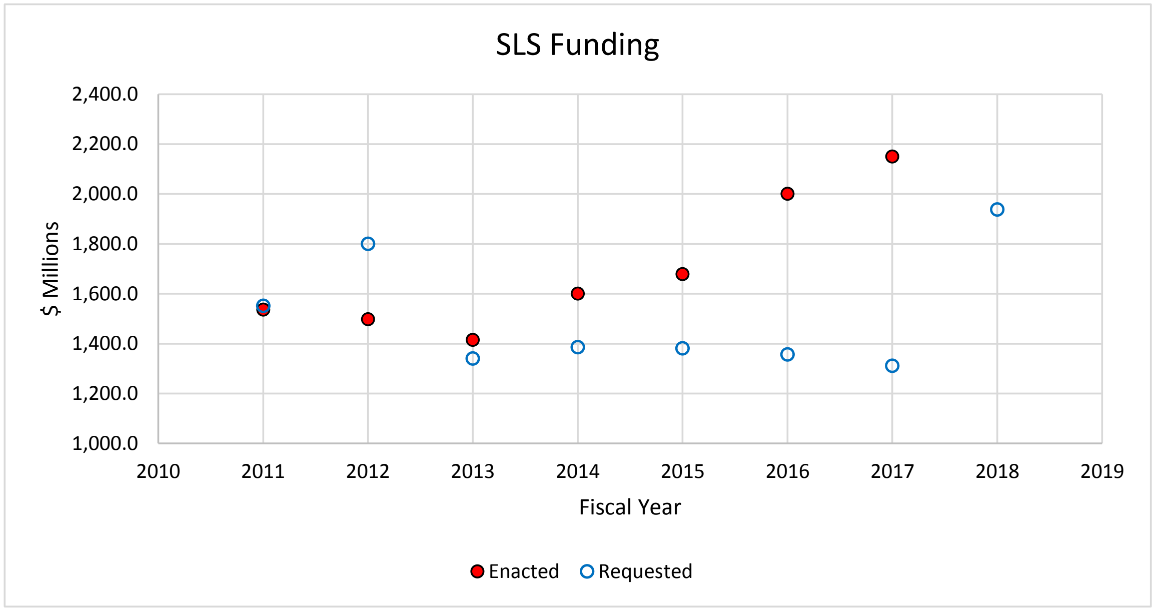


**Figure 3:** SLS program funding has seen instability similar to that experienced by the Constellation program. Polarization between the White House and Congress resulted in early budgetary cuts which are now manifesting in cost overruns and schedule slips in the later years of development [79]. This is similar to the phenomenon described in Figure 1.

November 2018, and is in the process of being rescheduled again for 2019 [1, 2, 3, 112]. NASA, under the Flexible Path Option, has been directed to capture and explore a Near Earth Asteroid in an effort to develop technologies that will be needed for future Mars missions [83].

However, in the mid second decade, rejuvenated interest in lunar exploration by other international partners prompted yet another shift in policy. The NASA Authorization Act of 2017 directed NASA to reevaluate the value of the Asteroid Redirect Mission to capture and explore a near Earth asteroid, while also putting a new emphasis on a mission to Jupiter's moon, Europa, along with increased cis-lunar operations[3]. Other legislation also proposes redirecting NASA to return to a Moon first option in preparation for Mars surface exploration [116]. This continual redirection, long-lasting uncertainty, and variability that has plagued U.S. manned space exploration policies of the early 21st century has produced challenges in designing and developing the systems and architectures that are required for manned deep space exploration missions.

## 1.2 Design in a Time of Uncertainty

The task of designing complex architectures is by no means trivial. To do so with such political instability and uncertainty, which trickles into the programmatics, presents increased challenges to the designer. This section will discuss relevant terminology, the process of design, and highlight specific challenges that arise from such uncertainty.

### 1.2.1 Phases of Design

The process of design and its corresponding phases are applied in a wide variety of disciplines. Each of these disciplines provides its own view as to what phases should exist in the process and what milestones mark the boundaries between those phases. Typically, in the aircraft industry, design is described as having three major phases:

conceptual design, preliminary design, and detailed design [94]. From a systems engineering viewpoint, Arthur Hall describes the phases as: system studies, exploratory planning, development planning, studies during development, and concurrent engineering [41]. David Ullman uses similar phases when describing his mechanical design process: project definition and planning, specification definition, conceptual design, product development, and product support [109].

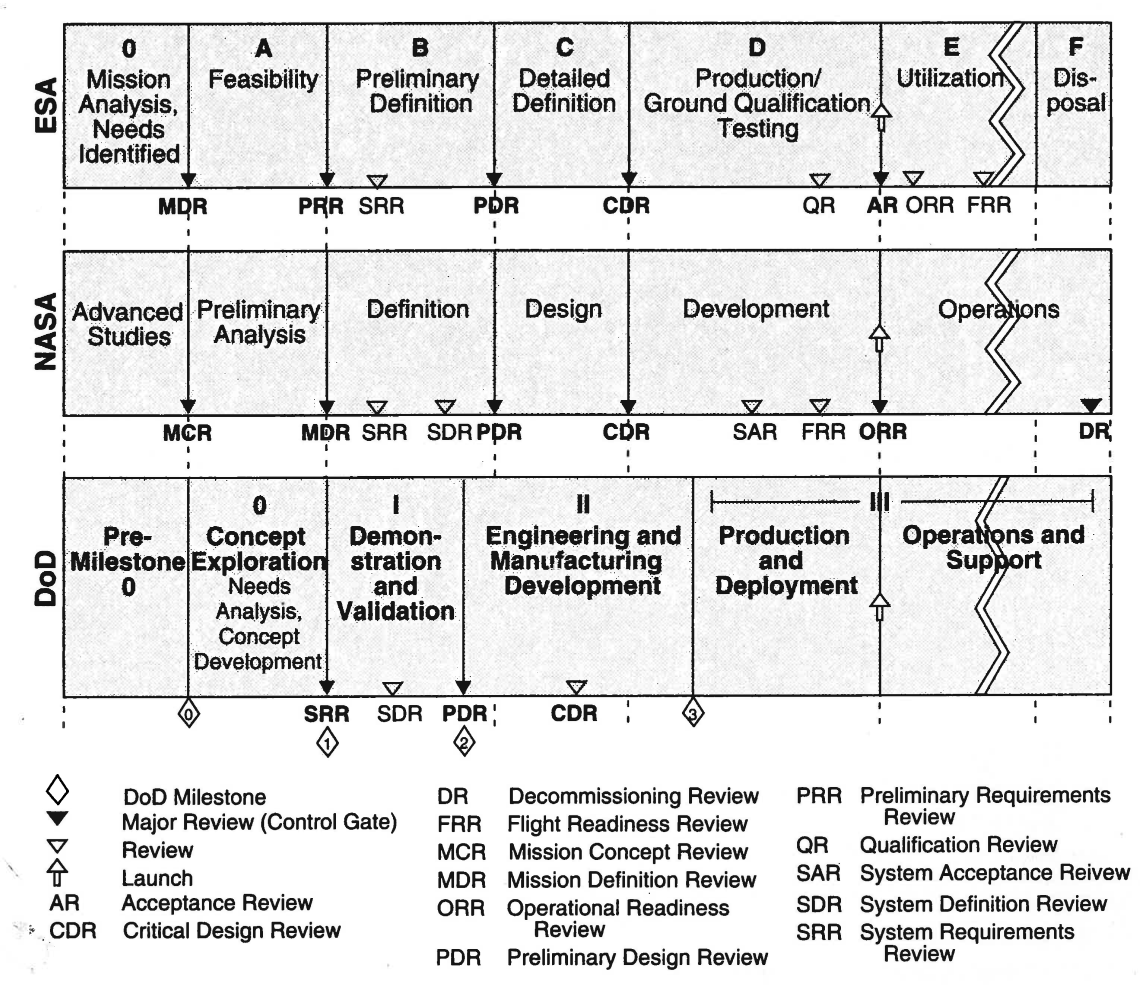
Focusing on the space industry, some of the key stakeholders include the U.S. Department of Defense (DoD), the European Space Agency (ESA) and the National Aeronautics and Space Administration (NASA). A mapping of their respective phases of design is shown in Figure 4 [63]. This figure highlights the commonality in the phases of design as described by these key stakeholders. The boundaries between the phases are reviewed at the bottom of each process. The individual stakeholders each have their own set of reviews, but there are three they all agree on: the system requirements review (SRR), the preliminary design review (PDR), and the critical design review (CDR).

Because the motivational problem for this dissertation is focused on space architectures, the phases of design are defined using NASA's systems engineering conventions. NASA defines seven life-cycle phases in the design process ranging from conceptual studies all the way through operation and closeout. Figure 5 depicts the sequence of the seven phases along with the associated milestones [82]. The following is a brief description of the seven phases [77].

Pre-Phase A: Concept Studies – Devise various feasible concepts from which new projects and programs can be selected.

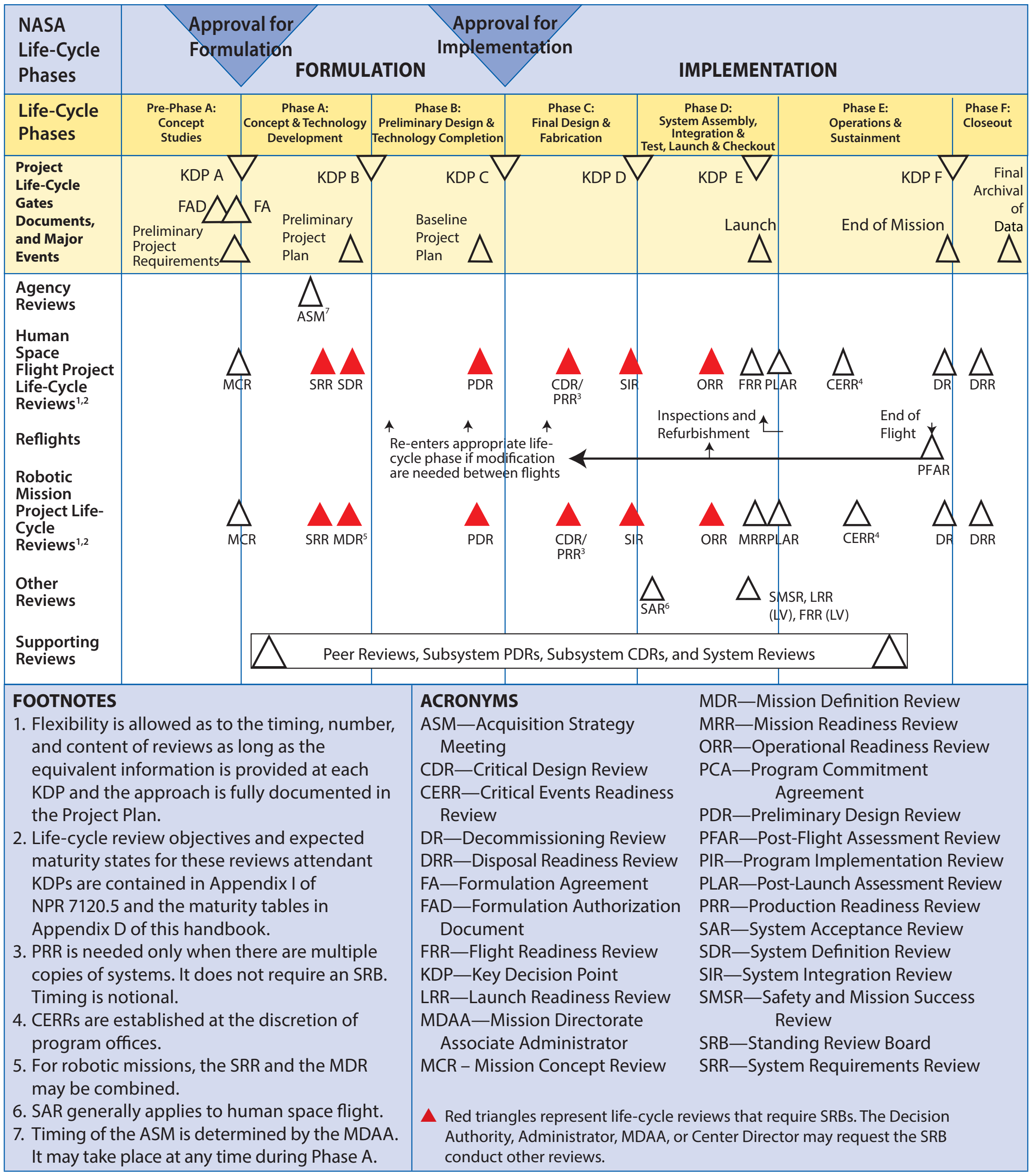
Phase A: Concept and Technology Development – Fully develop a baseline mission concept and begin or assume responsibility for the development of needed technologies.

Phase B: Preliminary Design and Technology Completion – Establish an initial project



ESA
0 Mission Analysis, Needs Identified
A Feasibility
B Preliminary Definition
C Detailed Definition
D Production/ Ground Qualification Testing
E Utilization
F Dis-posal
MDR PRR SRR PDR CDR QR AR ORR FRR
NASA
Advanced Studies
Preliminary Analysis
Definition
Design
Development
Operations
MCR MDR SRR SDR PDR CDR SAR FRR ORR DR
DoD
Pre-Milestone 0
0 Concept Exploration Needs Analysis, Concept Development
I Demon-stration and Validation
II Engineering and Manufacturing Development
III
Production and Deployment
Operations and Support
SRR SDR PDR CDR
0
1
2
3
DoD Milestone
Major Review (Control Gate)
Review
Launch
AR Acceptance Review
CDR Critical Design Review
DR Decommissioning Review
FRR Flight Readiness Review
MCR Mission Concept Review
MDR Mission Definition Review
ORR Operational Readiness Review
PDR Preliminary Design Review
PRR Preliminary Requirements Review
QR Qualification Review
SAR System Acceptance Reivew
SDR System Definition Review
SRR System Requirements Review

**Figure 4:** The phases of design for various space industry stakeholders have a high degree of commonality. Critical reviews throughout the design processes are shown below each respective process. [63].


NASA Life-Cycle Phases
Approval for Formulation
FORMULATION
Approval for Implementation
IMPLEMENTATION
Life-Cycle Phases
Pre-Phase A: Concept Studies
Phase A: Concept & Technology Development
Phase B: Preliminary Design & Technology Completion
Phase C: Final Design & Fabrication
Phase D: System Assembly, Integration & Test, Launch & Checkout
Phase E: Operations & Sustainment
Phase F: Closeout
Project Life-Cycle Gates Documents, and Major Events
KDP A
FAD
FA
Preliminary Project Requirements
KDP B
Preliminary Project Plan
KDP C
Baseline Project Plan
KDP D
KDP E
Launch
KDP F
End of Mission
Final Archival of Data
Agency Reviews
ASM[7]
Human Space Flight Project Life-Cycle Reviews[1,2]
MCR
SRR SDR
PDR
CDR/ PRR[3]
SIR
ORR
FRR PLAR
CERR[4]
DR
DRR
Reflights
Re-enters appropriate life-cycle phase if modification are needed between flights
Inspections and Refurbishment
End of Flight
PFAR
Robotic Mission Project Life-Cycle Reviews[1,2]
MCR
SRR MDR[5]
PDR
CDR/ PRR[3]
SIR
ORR
MRR PLAR
CERR[4]
DR
DRR
Other Reviews
SAR[6]
SMSR, LRR (LV), FRR (LV)
Supporting Reviews
Peer Reviews, Subsystem PDRs, Subsystem CDRs, and System Reviews

**FOOTNOTES**

1. Flexibility is allowed as to the timing, number, and content of reviews as long as the equivalent information is provided at each KDP and the approach is fully documented in the Project Plan.
2. Life-cycle review objectives and expected maturity states for these reviews attendant KDPs are contained in Appendix I of NPR 7120.5 and the maturity tables in Appendix D of this handbook.
3. PRR is needed only when there are multiple copies of systems. It does not require an SRB. Timing is notional.
4. CERRs are established at the discretion of program offices.
5. For robotic missions, the SRR and the MDR may be combined.
6. SAR generally applies to human space flight.
7. Timing of the ASM is determined by the MDAA. It may take place at any time during Phase A.

**ACRONYMS**

| | |
|---|---|
| ASM | Acquisition Strategy Meeting |
| CDR | Critical Design Review |
| CERR | Critical Events Readiness Review |
| DR | Decommissioning Review |
| DRR | Disposal Readiness Review |
| FA | Formulation Agreement |
| FAD | Formulation Authorization Document |
| FRR | Flight Readiness Review |
| KDP | Key Decision Point |
| LRR | Launch Readiness Review |
| MDAA | Mission Directorate Associate Administrator |
| MCR | Mission Concept Review |
| MDR | Mission Definition Review |
| MRR | Mission Readiness Review |
| ORR | Operational Readiness Review |
| PCA | Program Commitment Agreement |
| PDR | Preliminary Design Review |
| PFAR | Post-Flight Assessment Review |
| PIR | Program Implementation Review |
| PLAR | Post-Launch Assessment Review |
| PRR | Production Readiness Review |
| SAR | System Acceptance Review |
| SDR | System Definition Review |
| SIR | System Integration Review |
| SMSR | Safety and Mission Success Review |
| SRB | Standing Review Board |
| SRR | System Requirements Review |

🔺 Red triangles represent life-cycle reviews that require SRBs. The Decision Authority, Administrator, MDAA, or Center Director may request the SRB conduct other reviews.

**Figure 5:** The NASA Project Life Cycle is fundamentally divided between formulation and implementation. The formulation consist of a preparatory Pre-Phase A, followed by Phase A and Phase B. Implementation consists of Phases C,D,E, and F. Though both the manned spaceflight and robotic communities have developed slightly different terms and launch approval processes, the project management life cycles are essentially the same [82].

baseline such that system and subsystem-level specification can be derived from project-level requirements.

Phase C: Final Design and Fabrication – Establish a complete design, fabricate or produce hardware, and code software in preparation for integration.

Phase D: System Assembly, Integration and Test, Launch – Assembly, integration, verification, and validation of the system, including testing the system to expected environments.

Phase E: Operations and Sustainment – Conduct the prime mission and meet the initially identified need and maintain support for that need.

Phase F: Closeout – Implement system decommissioning disposal planning and analyze returned data and/or samples.

### 1.2.2 Design Freedom versus Design Knowledge

Through the course of the design phases, there are several concepts that must be weighed: cost, design knowledge, and design freedom. Contrary to traditional thought, cost is not incurred at the time committed, but rather through the process of making design decisions [27]. Decisions about the design tend to be made early in the design process. This means that a majority of the cost for a design is committed very early in the design process, while that cost is not incurred until later in the design process. Many studies have examined this behavior. One such study determined that only 20% of the cost is incurred during the early phases of design, while those same phases commit 80% of the cost [24]. Figure 6 illustrates this relationship between cost, ease of change, and design knowledge. Here, ease of change can be interpreted as a measure of design freedom.

Typically, design decisions are made early in the design process, when knowledge is relatively low. This can result in uninformed decisions that can lead to costly design revisions in later phases, particularly during testing. A good example of this

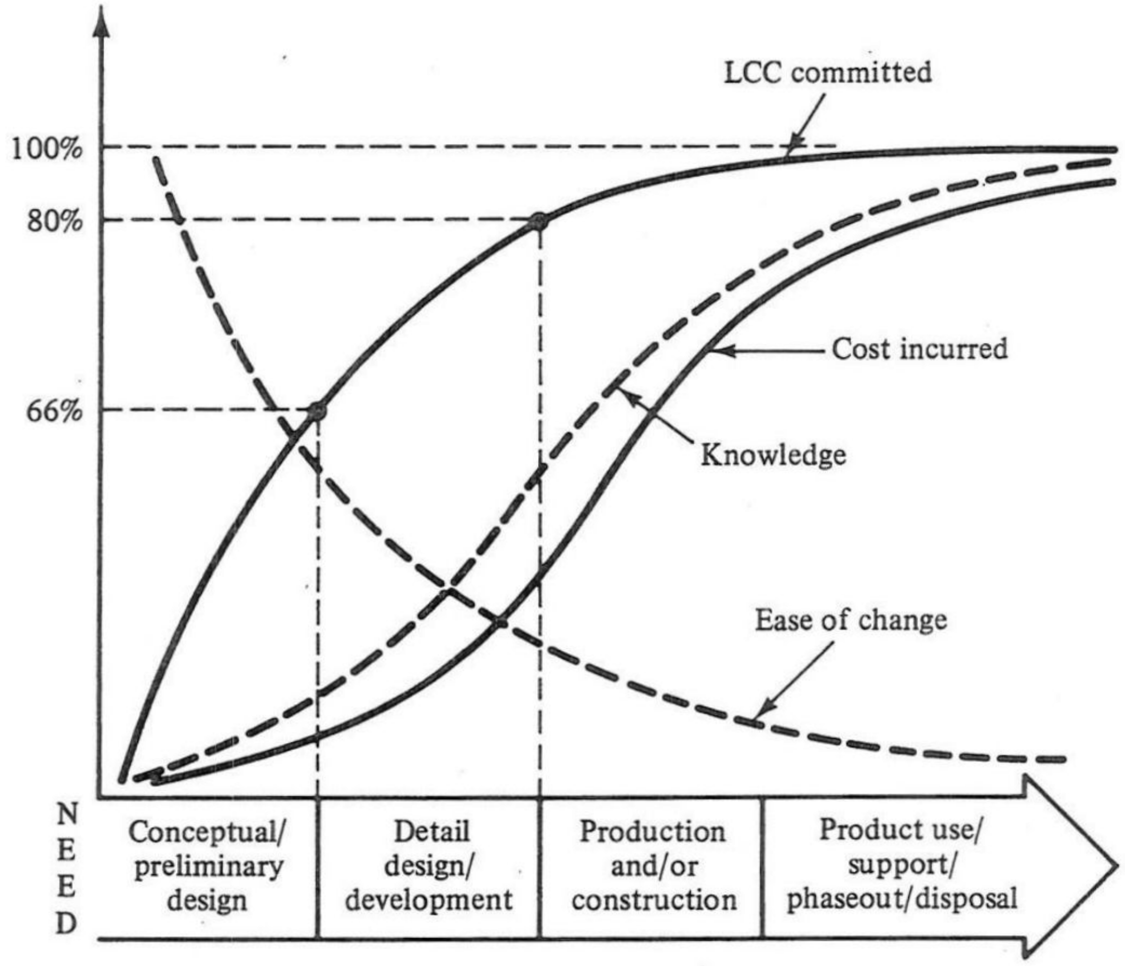


**Figure 6:** Cost, design knowledge, and ease of change (design freedom), are all related in the design process. Contrary to traditional thought, much of the cost of a design is committed much earlier than it is incurred. This is due to design decisions made early in the design process. Cost committed and design freedom are inversely related. Bringing design knowledge forward in the design process will maintain a higher design freedom longer into the design process, ultimately reducing cost and schedule overruns. [27].

behavior is in the development of liquid rocket engines. Glen Havskjold performed a study on historical development programs from the Pratt & Whitney Rocketdyne Company [44, 45, 46]. Due to a lack of design knowledge during the initial design phase of these engines, a pattern of test-fail-fix occurs during the development and testing. The result is increased costs and schedule of the engines studied. In fact, 73% of the development cost of the F-1, J-2, and Space Shuttle Main Engine were determined to be due to corrective actions during full-scale testing [44].

These relationships between design knowledge, design freedom, and cost, indicate the need for well-informed decisions early in the design process. These decisions are vital to reducing the risk of increased cost and schedule due to design iterations [34]. Industry and academia have been working towards this goal through various means [93, 22, 105, 68]. These methods share similar techniques of bringing design knowledge earlier into the design process in an attempt to maintain design freedom longer while allowing decision makers to make informed decisions about the design, leading to reduced cost and schedule.

### 1.2.3 Challenges

Attempting to design an architecture in an environment of uncertainty can present many challenges to the designer. This uncertainty can manifest in many forms, such as changing budgetary constraints, capabilities and requirements creep, and changing mission [110, 111]. Designers are forced to consider increasingly complex architectures with growing design spaces to mitigate issues that may arise from these uncertainties. However, the growth in the architecture space presents new problems in the form of overruns due to an increased lack of knowledge and the sheer number of alternatives that exist to fully define the architecture design space. In their work on model-based systems engineering, Jon Holt and Simon Perry claim that projects fail due to complexity, lack of understanding, and communication issues [49]. Furthermore, these underlying reasons do not exist in isolation, but feed upon each other. Complexity issues will lead to lack of understanding and communication problems. A lack of understanding will lead to communication problems and complexity. Finally, a breakdown in communication will result in a lack of understanding and unforeseen complexity.

#### *1.2.3.1 Cost and Schedule Overrun*

As was described in this chapter, political uncertainties tend to drive fluctuations in defined missions, goals, and funding profiles. This constant flux forces designers to consider ever-growing architecture spaces which become difficult to fully define and understand. This lack of knowledge of the architecture space has been shown to drive both cost and schedule overruns in the projects associated with these architectures. Additionally, incorporation of new technologies has been shown to have a dramatic impact on the overall cost of programs. Depending on the maturity of incorporated technologies, costs for a given system can vary by as much as 50%, while total program costs may grow exponentially with the time to develop a technology [64]. These observations further support the importance of understanding architecture spaces early in the design process to account for these potential cost growths and uncertainties.

The initial phases of the design process should capture a large amount of design knowledge to help mitigate risks associated with such uncertainty. With this, decision makers are able to make more informed decisions that can reduce the risk of costly design iterations when traceable, quantitative information is provided. In contrast, qualitative data can fail to capture trends which may exist that could inform the decision maker in preventing cost and schedule overruns [62]. This leads to a need for an intelligent, methodical, comprehensive exploration of the architecture space that is quantitative in nature.

#### *1.2.3.2 Combinatorial Explosion*

The encyclopedia of Operations Research and Management Science defines combinatorial explosion as the phenomenon associated with optimization problems whose computational difficulty increases exponentially with the size of the problem [35]. The uncertainties stated earlier that result in these large, complex architecture spaces will also result in challenges with regard to physical computation and analysis of the design space.

Peter Schuster claims that combinatorial explosion is a result of assembling objects from elements by means of predefined combination rules [102]. If architectures consist of a collection of elements combined to meet a given objective, the design space defined by the various combinations of these elements is likely to suffer from this notion of combinatorial explosion. In his development of a method to rapidly analyze architectures in an attempt to study complex architecture design spaces, Joseph Iacobucci provides context as to the scale of the number of alternatives that may exist in complex architecture spaces [50]. Even after considering compatibility constraints, this number can still be an impractical billions of alternatives to analyze. However, early phases of the design process may not require every possible combination of alternatives to be evaluated to achieve a drastic increase in knowledge of the architecture space. This leads to the second need of gaining an understanding of the architecture space, through analysis, that provides an increase in design knowledge which is sufficient such that decision makers are able make informed design decisions in the early design phases.

## 1.3 Statement of Purpose

The previously mentioned challenges and needs form the basis and driving force behind this body of work, which is stated as follows:

**Statement of Purpose**

To provide a capability to analyze complex systems of systems to an extent which will provide decision makers in the early phases of design sufficient information to reduce the risks associated with cost and schedule overruns due to lack of design knowledge.

## *1.4 Document Organization*

Chapter 2 provides background information relevant to this body of work. Section 2.1 provides clarification of terms typical in the military domain, but which have slight nuances within the context of space transportation. The methods discussed in Section 2.3, Section 2.4, and Section 2.5, along with the concepts presented in Section 3.3, will guide the development of research questions and hypotheses related to integrated architecture and technology exploration at a subsystem level. In order to perform the experiments designed to test these hypotheses in the domain motivating this work, new models are developed for space transportation architecting, summarized in Chapter 4. The research questions developed throughout Chapter 3 will be explored through experiments presented in Chapter 5, the results of which will be the basis for defining a formal framework for integrated architecture and technology exploration at a subsystem level, presented in Section 5.2, and implemented through a notional case study in Section 5.3. Finally, Chapter 6 summarizes this research, offering a set of contributions, as well as suggestions for future work.

*Note from the author: The officially published electronic version of this document contains extensive use of hyperlinks. The table of contents, list of figures, and list of tables link to their respective chapters, sections, figures, and tables throughout the document. Within the document, references are linked to their respective bibliographic entry information, and abbreviations are linked to their nomenclature definitions. Finally, a subject index is provided with active page links. This is to aid the reader in examining relevant information as needed. They are not physically visible due to guidelines imposed by the Georgia Institute of Technology regarding formatting of electronic dissertation documents.*

# CHAPTER II

# BACKGROUND

This chapter contains background information on existing techniques which represent possible candidates to address the statement of purpose presented in Chapter 1. Section 2.1 defines various terms used in the field of architecture design and analysis and how they are unique to the focus of this research. Section 2.3 and Section 2.4 present overviews on current methods that exist in architecture design and technology evaluation, respectively. The chapter concludes with a brief overview of Model-Based Systems Engineering and various developed languages in Section 2.5. The intent of this chapter is not to provide a comprehensive discussion of each technique, but rather to provide a brief description of the capabilities of each technique to an unfamiliar reader such that a discussion of gaps and needs may be discussed in Chapter 3.

## *2.1 Terminology*

It is important to take a moment to define the concepts that exist in the realm of space systems design. Many readers will be familiar with the terms system, system of systems, vehicle, mission, technology, architecture, and campaign. There may exist several accepted definitions for a term. In these instances, the implied definition for these terms throughout the remainder of this document shall be those presented in this section.

### 2.1.1 System

The Merriam-Webster Online Dictionary defines a system as, "A regularly interacting or interdependent group of items forming a unified whole" [106].

This general definition is intentionally vague to ensure it captures all possible cases that may be considered a system. However, for our purpose, a more detailed description of a system is desired. George Dieter defines a system as, "The entire combination of hardware, information, and people necessary to accomplish some specified mission" [22]. The International Council on Systems Engineering (INCOSE) definition for a system is: "A combination of interacting elements organized to achieve one [or] more stated purposes" and "An integrated set of elements, subsystems, or assemblies that accomplish a defined objective" [52]. These definitions help to clarify that the elements can be products, processes, people, information, techniques, facilities, services and other support elements.

The U.S. Department of Defense Architecture Framework (DoDAF) defines a system as, "A functionally, physically, and/or behaviorally related group of regularly interacting and interdependent elements" [54]. The same definition appears in the DoDAF v2 Manager's Guide [115]. ISO/IEC/IEEE 24765:2010 defines a system as a, "Combination of interacting elements organized to achieve one or more stated purposes" [51]. These definitions imply that these elements are not just randomly assembled, but are related and regularly interacting.

NASA defines a system as, "a construct or collection of different elements that together produce results not obtainable by the elements alone" [77]. This definition, along with many of the aforementioned definitions, have a common theme that the collection of elements is brought together for a purpose, mission or result otherwise unattainable by the individual elements [77, 22, 52, 51]. To summarize the key concepts of a system:

- A thoughtful, organized assembly of elements
- Regular interaction and interdependence between elements
- Elements can be products, processes, people, information, techniques, facilities, services, and other support elements

- Elements are brought together to achieve some stated purpose that is otherwise unattainable by the individual elements

For the purpose of this research, the word system shall mean an organized set of regularly interacting and interdependent products, processes, people, information, techniques, facilities, services, and other support elements, collectively known as subsystems, brought together for a stated purpose otherwise unattainable.

### 2.1.2 System of Systems

Literature provides a plethora of definitions for a system of systems (SoS). However, different fields have adopted their own, slightly tailored form. For the purpose of this dissertation, the following concepts and definitions will be considered.

ISO/IEC/IEEE 24765:2010 defines an SoS as, "A large system that delivers unique capabilities, formed by integrating independently useful systems" [51].

The Defense Acquisition Guidebook defines an SoS as, "a set or arrangement of systems that results when independent and useful systems are integrated into a larger system that delivers unique capabilities" [114].

Dimitri Mavris and Charles Dickerson define an SoS as, "a combination of interacting systems [i.e., elements of the SoS] integrated to realize properties, behaviors, and capabilities that achieve one or more stated purpose(s)" [21].

These definitions generally agree that a system of systems is a set of interacting independent systems, brought together to achieve unique capabilities. However, they fail to provide a scale that is typical of an SoS problem. The INCOSE defines an SoS as, "a system-of-interest whose system elements are themselves systems; typically, these entail large-scale interdisciplinary problems with multiple, heterogeneous, distributed systems" [52]. This definition provides insight into the problems that a system of systems typically aims to solve and their scale.

The U.S. Department of Defense Joint Capabilities Integration and Development

System (JCIDS) Manual defines an SoS as, "a set or arrangement of interdependent systems that are related or connected to provide a given capability. The loss of any part of the system will significantly degrade the performance or capabilities of the whole" [15]. The statement of impact on the performance capability of the whole if a component system fails is extremely relevant to space systems.

The previous definitions show general concepts that define a system of systems. Component systems within a system of systems have a level of independence from each other. The component systems are not necessarily defined by being included in a system of systems, but do work together for the purpose of the system of systems and are autonomous. Systems of systems typically aim to solve large-scale interdisciplinary problems, while Iacobucci states that the effects of a system of systems are often non-linear [50]. It should be noted, that though the component systems of an SoS are not necessarily defined by being a part of an SoS, space flight situations typically lead to component systems that are highly specialized and designed for a given SoS problem. This can be attributed to extremely unique operational environments, coupled with a high cost of access to space derived from labor-intensive designs [118]. Section 2.1.6 will define an architecture within the scope of this dissertation and will lead to a conclusion that an architecture defined by this dissertation is a system of systems.

### 2.1.3 Vehicle

NASA defines a vehicle as, "a structure, machine, or device, such as an aircraft or rocket, designed to carry a burden through air or space"[5]. Though old in origin, the definition as it pertains to this research is highly relevant. For the purpose of this dissertation, a vehicle shall be defined as "a structure, machine, or device designed to carry a burden." The burden in this definition is typically referred to as a payload. This payload can be an inert mass, such as another machine or device, as well as biological in nature, particularly, humans. It is an important distinction in the definition of a space vehicle because it implies very different design drivers despite potentially similar goals.

The term spacecraft is typically used to denote a vehicle designed specifically for use in space and may be used interchangeably with vehicle in this dissertation. Table 1 lists the traditional spacecraft elements or subsystems. Figure 7 shows the interdependence of the spacecraft subsystems for an unmanned vehicle. On a man-rated spacecraft, environmental control and life support systems (ECLSS) would have dependencies on the power and thermal control. The interdependencies of the spacecraft subsystems leads to the conclusion that a spacecraft fits the definition of a system, as given in Section 2.1.1 and is merely a special case of a system applied to space travel.

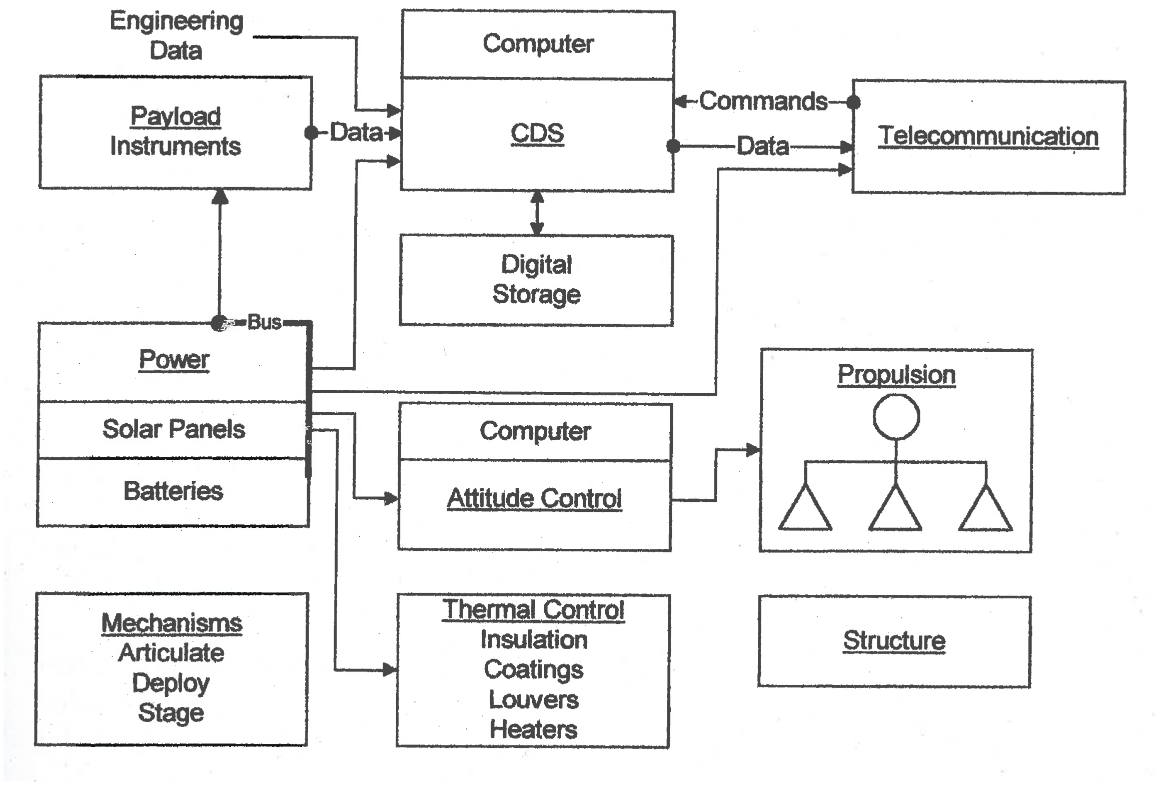


**Figure 7:** A simplified notional spacecraft block diagram showing the interdependence of various spacecraft elements for an unmanned vehicle [12].

**Table 1:** Common Spacecraft Elements [118, 63, 12]

| **Element Name** | **Function** |
|---|---|
| Propulsion | Spacecraft thrust, including fuel storage and plumbing |
| Attitude Determination and Control (ADC) or Attitude Control System (ACS) | Sensors, actuators, and software necessary to control the spacecraft orientation |
| Position and Orbit Determination and Control | Sensors and software necessary to control the spacecraft orbit |
| On Board Processing or Command and Data Handling (C&DH) | Electronics and software used to receive and distribute commands and to store and forward payload data and spacecraft telemetry |
| Telemetry, Tracking, and Command (TT&C) or RF Communications | Radio and associated hardware, such as cabling and antennas, used to communicate with the ground or other spacecraft |
| Power | Electronics, power generation, and power storage devices, as well as harnessing for power distribution |
| Structures and Mechanisms | All the hardware that supports the spacecraft, including the primary structural components, brackets, fasteners, and the actuators and mechanisms associated with deployed or movable structures |
| Thermal Control | All of the hardware necessary to control the temperature of the spacecraft |
| Environmental Control and Life Support System (ECLSS) **Manned Only** | All of the hardware necessary to support human life on board the spacecraft |
| Extra Vehicular Activity (EVA) Support and Robotics **Manned Only** | All of the hardware necessary to support human and robotic operations outside the spacecraft |

### 2.1.4 Mission

The Merriam-Webster Online Dictionary defines a mission as, "A definite military, naval, or aerospace task" [75]. Again, this definition is too general and vague for the purpose of this research. The U.S. Department of Defense clarifies by defining a mission as, "The task, together with the purpose, that clearly indicates the action to be taken and the reason therefore" [54, 55]. This definition clarifies that a mission is not just an arbitrary task, but has a purpose and reason coupled with required actions.

Charles Brown states that space missions provide seven classes of services [12]:

- Communication
- Navigation
- Weather
- Earth Resources
- Astronomy
- Planetary Exploration
- Manned Spacecraft

From the definition provided by the DoD, the purpose of a mission falls into one of the seven service classes listed above. The specific tasks are typically described by the orbits, trajectories, maneuvers, and operations required to achieve the given purpose.

### 2.1.5 Technology

The Merriam-Webster Online Dictionary defines a technology as, "The practical application of knowledge especially in a particular area" [107]. In the field of engineering, this application is typically to develop something new in an effort to achieve some goal. NASA defines a technology as, "A solution that arises from applying the discipline of engineering science to synthesize a device, process, or subsystem, to enable a specific capability" [85]. NASA expands the definition further to include processes and methods, as well as tangible hardware. However, the end goal is the same, to enable a specific capability, presumably one that could not be achieved before, or in a more efficient manner than before. The definition as presented by NASA shall be the implied meaning of the term "technology" in this body of work. This definition has the implication that a technology is something at the device or subsystem level. In the discussion to follow in Chapter 3, technology evaluation will be integral in fully understanding the design space. As such, a scale on which to describe technologies is crucial.

**Table 2:** NASA Technology Readiness Level Scale [77]

| **TRL** | **Definition** |
|---|---|
| 9 | Actual system "flight proven" through successful mission operations |
| 8 | Actual system completed and "flight qualified" through test and demonstration (ground or flight) |
| 7 | System prototype demonstration in a target/space environment |
| 6 | System/subsystem model or prototype demonstration in a relevant environment (ground or space) |
| 5 | Component and/or breadboard validation in relevant environment |
| 4 | Component and/or breadboard validation in laboratory environment |
| 3 | Analytical and experimental critical function and/or characteristic proof-of-concept |
| 2 | Technology concept and/or application formulated |
| 1 | Basic principles observed and reported |

NASA developed the concept of the technology readiness level (TRL) scale in the 1970s as a tool for assessing the maturity of technologies during complex system

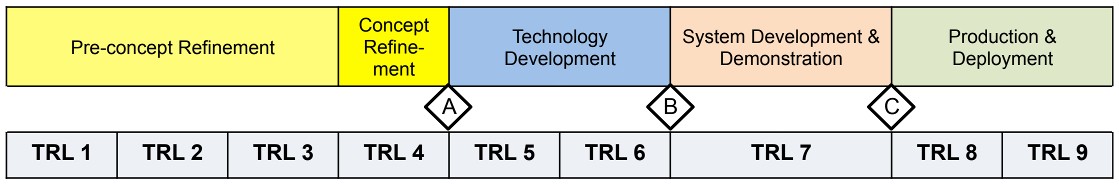


**Figure 8:** Mapping of technology readiness levels to U.S. Department of Defense System Acquisition Process. Technologies are expected to achieve TRL 4 by milestone A, TRL 6 by milestone B, and TRL 7 by milestone C [88, 7].

development. Table 2 lists the state at which a technology is considered to be a specific TRL. Since the scale's inception, organizations are increasingly mapping TRL to their own systems development processes. For example, the U.S. Department of Defense mapped TRL to their own System Acquisition Process, as shown in Figure 8. This is done primarily for its shared understanding of technology maturity and risk [88], not to create a shared common definition of technology across these two domains.

### 2.1.6 Architecture

The Merriam-Webster Online Dictionary defines an architecture as, "A unifying or coherent form or structure" [6]. While the DoDAF defines an architecture as, "A framework or structure that portrays relationships among all the elements of the subject force, system or activity" [115]. From this, an architecture is known to have structure and relationship among the constituent components.

ISO/IEC/IEEE 24765:2010 defines an architecture as a, "Fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution" [51]. Though this definition was developed with context to electronics, it is still applicable here. This definition broadens the relationship aspect of an architecture to not only include the constituent components, but also with its surrounding environment. Environments can have a drastic impact on the design of a system or SoS, especially in space applications.

Mavris and Dickerson build upon this by defining an architecture as, “The fundamental organization of a system, embodied in its components, their relationships to each other and the environment, the principles governing its design and evolution, its purpose, and its attractiveness” [21]. Here, it is stated that the organization of components has a purpose and a certain attractiveness that can distinguish it from another organization of components.

For this dissertation, the definition of an architecture shall be that presented by Mavris and Dickerson, as well as ISO/IEC/IEEE 24765:2010. The definition makes no distinction on the form of the system, and as such, may in fact be a system of systems as defined in Section 2.1.2. This would result in the components being systems themselves. This link implies that an architecture may be an instance of a system of systems.

Applying this definition to the realm of space vehicles and missions, a space architecture can be described as having the elements seen in Table 3. Larson and Pranke state that a mission concept, along with the functional and physical elements defined by this concept, form the basis of the space architecture [12]. The body of this work will focus on exploring the architecture design space, with emphasis on orbits and trajectories, space elements, and surface elements. Further details regarding the specific research objectives can be found in Section 3.2

### 2.1.7 Campaign

The Merriam-Webster Online Dictionary defines a campaign as, “A connected series of operations designed to bring about a particular result” [14].

The U.S. Department of Defense defines a campaign as, “A series of related major operations aimed at achieving strategic and operational objectives within a given time and space” [54].

This body of research will consider the definition provided by the DoD to define

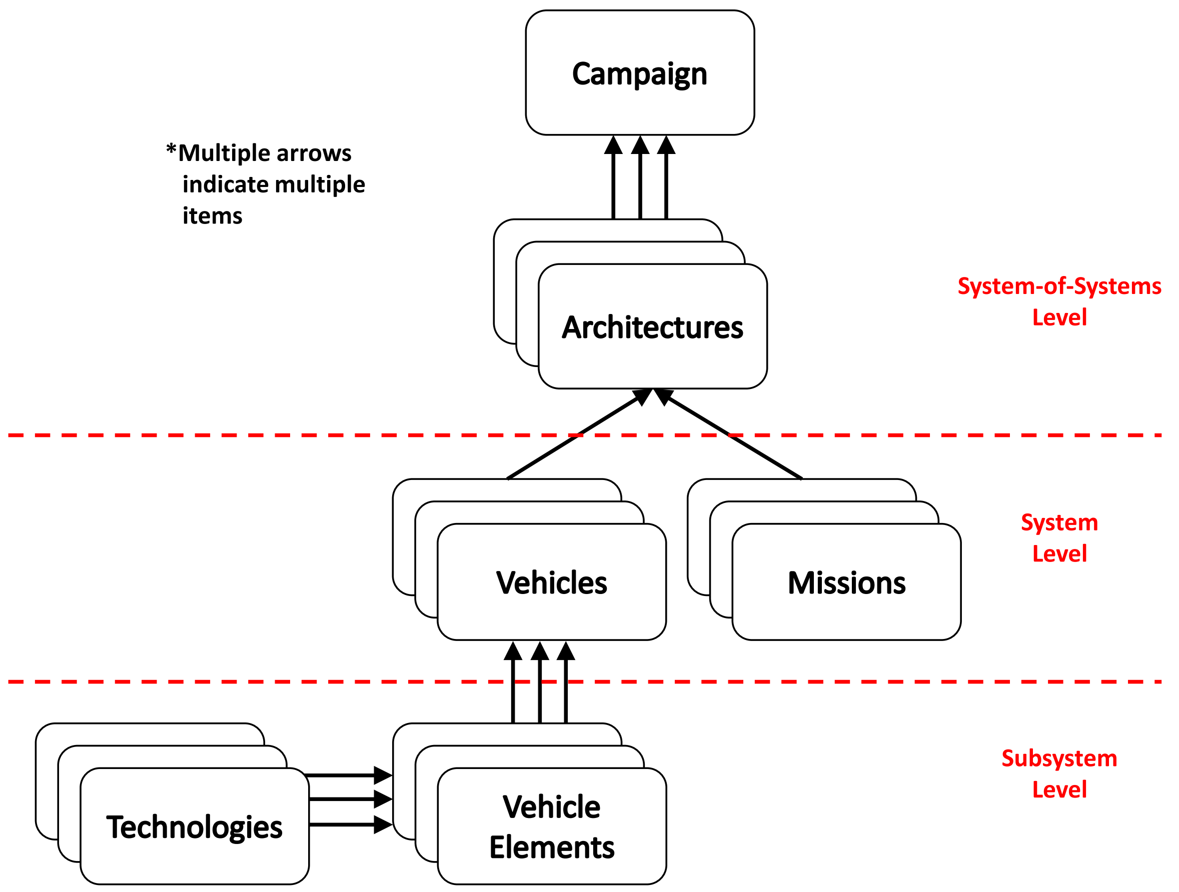


**Figure 9:** The structure of a campaign as defined includes the various items described in Section 2.1. Campaigns consist of multiple architectures. An architecture is a mission and the functional and physical elements it defines. The functional elements of interest for this research are the orbits and trajectories. The physical elements of interest for this research are the space elements, and surface elements. The space elements are typically the vehicles, while the surface elements are typically the payloads. Technologies are applied to the subsystems of the space and surface elements.

**Table 3:** Common Space Architecture Elements [63]

| **Element Name** | **Examples** |
|---|---|
| Operations Elements | • Communication Operations Concepts<br>• Operations Functions<br>• Space Logistics<br>• Command, Control, and Communication |
| Orbits and Trajectories | • Earth Orbits<br>• Interplanetary Transfers<br>• Planetary Orbits<br>• Entry, Descent, Landing, and Ascent |
| Transportation Elements | • Earth-to-Orbit Vehicle<br>• Launch Facilities |
| Space Elements | • In-Space Vehicle<br>• Vehicle for Entry, Descent, Landing, and Ascent |
| Surface Elements | • Surface Bases<br>• Surface Vehicles<br>• In-Situ Resources |
| Crew<br>**Manned Only** | • People as Payload or Operators<br>• Physiology and Psychology<br>• Human Factors<br>• Safety and Reliability |

a campaign, specifically focusing on a context to space. An example of a strategic or operational objective would be the exploration of the Martian surface, or the construction of a Lunar outpost. The related major operations shall be architectures as described in Section 2.1.6. To achieve the strategic or operational objective would require multiple architectures. An architecture within the context of this dissertation is described as a combination of functional and physical elements defined by a specific mission. As a result, this dissertation will consider a campaign as a set of missions and their related functional and physical elements brought together to achieve an overarching objective. Figure 9 provides a visual description of the structure of a campaign, and how the terms defined in this section are related to each other for the purpose of this research.

## *2.2 Desired Method Features*

In order to bring additional knowledge earlier into the design process, it would be desirable to observe both architectures and technologies together. This allows the effects due to incorporating technologies into architecture designs to be better understood. It is known that technologies have a dramatic impact on the overall cost and development schedule of designs [62, 99, 64]. This implies a need to consider technologies alongside architecture design during the conceptual design process to aid in increasing knowledge to reduce cost and schedule overruns. A method or technique should thereby include the ability to evaluate both architectures and technologies. Due to the definition of architectures and technologies in this dissertation at the subsystem level, identified methods and techniques should be capable of defining the problem at a subsystem level.

Furthermore, during the conceptual design process, considering a wide range of design alternatives and technologies will help to better understand the trade space. This will allow decision makers to make well-informed decisions early on during the design process to mitigate unnecessary cost growth and schedule slips. A method should be capable of evaluating many different architectures and technologies together to provide this information. Most real world decisions will present themselves in a multi-objective form, having multiple competing desires, such as cost and performance. The number of objectives to consider can vary dramatically depending on the goals of a given study. A method should be flexible enough to allow these varying objectives to be evaluated across both architectures and technologies. Table 4 summarizes the desired features discussed above.

The remainder of this chapter will focus on discussing modern, relevant methods and frameworks for the purposes of architecture design and technology evaluation. The ability of each method to meet the features outlined above will be detailed.

**Table 4:** Required Features for an Integrated Architecture Analysis and Technology Evaluation Framework

| **Feature** | **Purpose** |
|---|---|
| Architectures Defined at Subsystem Level | To enable effects due to technologies defined at the subsystem level to be evaluated |
| Can Evaluate Multiple Architectures | To enable large numbers of architecture alternatives to be evaluated simultaneously within the same trade space |
| Multi-objective Architecture Analysis | To enable architectures to be compared against a wide range of customer objectives such as cost, performance, reliability, risk, etc. |
| Technologies Defined at Subsystem Level | To enable subsystem or component level solutions to technical challenges within a design |
| Can Evaluate Multiple Technologies | To enable large numbers of technologies to be considered simultaneously within the same trade space |
| Multi-objective Technology Evaluation | To enable technologies to be compared against a wide range of customer objectives, such as cost, performance, reliability, risk, etc., which may be propagated into the architecture analysis process |

## *2.3 Architecture Design Methods*

Recent research is attempting to mitigate the problem of combinatorial explosion in the design of complex architectures by providing methods capable of quickly analyzing the design space through novel methods. In the domain of system of systems architectures, alternatives are typically a set of discrete design decisions that result in a unique architecture. This discrete nature creates challenges with regard to evaluating and analyzing the architecture space. The evaluation frameworks of the following methods are of particular interest and relevance to this dissertation and will be highlighted.

### 2.3.1 ARCHITECT

The Architecture-Based Technology Evaluation and Capability Tradeoff (ARCHITECT) method proposes using architectures to describe the system of systems design space and to generate an architectural alternative space from it [39]. The alternatives are subsequently evaluated in order to generate data from which decision makers can gain information and insight.

ARCHITECT utilizes various advanced methods to define, evaluate, and assess the architecture design space. The Relational Oriented Systems Engineering and Technology Tradeoff Analysis (ROSETTA) environment provides a framework to allow decomposition and mapping of the architecture elements to functional and/or physical requirements [70, 20, 69]. Once analysis is performed, gaps in the current architecture are identified and the process of describing the alternative space begins. This is done by utilizing the Technique for the Enumeration of System of Systems Alternatives (TESSA), The Rapid Architecture Alternative Modeling (RAAM) framework, and the Architecture Resource-based Collaborative Network Evaluation Tool (ARCNET). Combining these methods allows the full set of architectural alternatives across all the defined dimensions to be described. TESSA provides a means of generating possible tasks, process flows, candidate systems, interface requirements, and consider organizational constraints [38]. RAAM provides the ability to combine the system and tasks to create the full set of system portfolios and operational implementations of each of those system portfolios [50]. ARCNET adds the full set of interface alternatives and force structures for the alternatives generated by RAAM [23]. Figure 10 provides the process flow diagram of the ARCHITECT method and how each step of the process maps to a typical engineering process.

ARCHITECT is capable of handling large numbers of alternatives, incorporating both architectures and technologies, to be evaluated on a multi-objective basis. However, a key disadvantage to ARCHITECT with regard to the features required of a

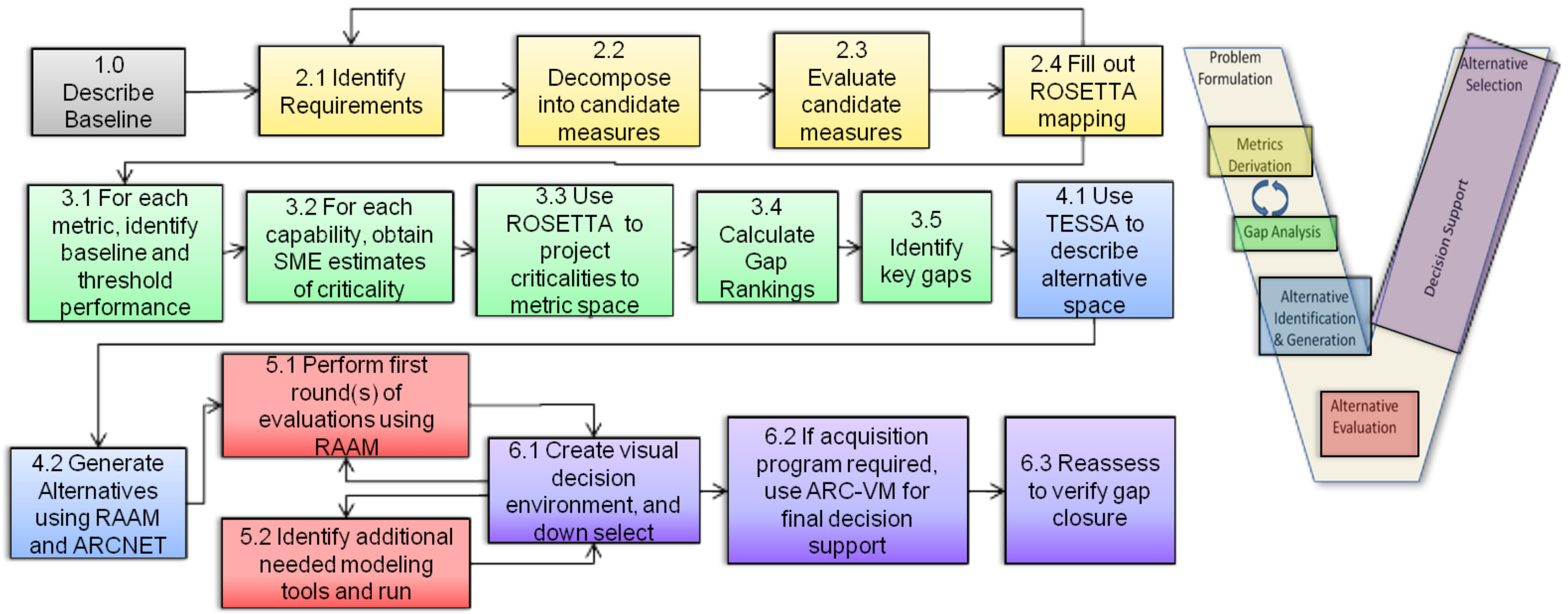

1.0 Describe Baseline
2.1 Identify Requirements
2.2 Decompose into candidate measures
2.3 Evaluate candidate measures
2.4 Fill out ROSETTA mapping
3.1 For each metric, identify baseline and threshold performance
3.2 For each capability, obtain SME estimates of criticality
3.3 Use ROSETTA to project criticalities to metric space
3.4 Calculate Gap Rankings
3.5 Identify key gaps
4.1 Use TESSA to describe alternative space
4.2 Generate Alternatives using RAAM and ARCNET
5.1 Perform first round(s) of evaluations using RAAM
5.2 Identify additional needed modeling tools and run
6.1 Create visual decision environment, and down select
6.2 If acquisition program required, use ARC-VM for final decision support
6.3 Reassess to verify gap closure
Problem Formulation
Metrics Derivation
Gap Analysis
Alternative Identification & Generation
Alternative Evaluation
Decision Support
Alternative Selection

**Figure 10:** ARCHITECT utilizes various methods in an effort to describe system of systems design spaces by using architectures [39]. These methods include ROSETTA [70, 20, 69] for the systems definition phase, and TESSA [38], ARCNET [23], and RAAM [50] for the alternatives generation and evaluation phases. ARCHITECT takes strong influence from the military domain, and as such, provides an operations-based approach to solving the system of systems problem.

method or framework for this research is the level of definition of architectures and technologies. ARCHITECT is formulated under the notion of system-level modeling of architecture elements. This does not meet the needs of a subsystem-level breakdown of an architecture, which is needed to meet the desire to evaluate subsystem-level technologies concurrently with architectures.

#### *2.3.1.1 RAAM*

RAAM provides the physical means by which systems and tasks are combined and evaluated in the ARCHITECT method. ARCHITECT aims at performing a full factorial analysis of the architecture space. RAAM enables this by providing a simple, lightweight definition of the input architectures such that the analysis does not require lengthy computation for any single architecture. This allows a great number of architectures to be evaluated in a short period of time.

RAAM receives the inputs of the required capabilities of an architecture from ARCHITECT where the capabilities are combined with the required tasks to create a full capability hierarchy. RAAM is flexible enough to work with a variety of computer models to perform the system of systems architecture analysis. Outputs of RAAM are combined into "portfolios" of architectures. These portfolios can contain architectures with the same physical system portfolio, as was shown in the canonical example found in Iacobucci's development of RAAM [50]. The full RAAM process is depicted in Figure 11.

### 2.3.2 STASE

The Set Theory-Influenced Architecture Space Exploration (STASE) method takes a different approach to architecture space evaluation. Where ARCHITECT performs a full factorial analysis of the architecture alternative space, STASE attempts to reduce the number of alternatives that are analyzed by utilizing set theory to define the system of systems problem. Figure 12 provides a visual representation of the

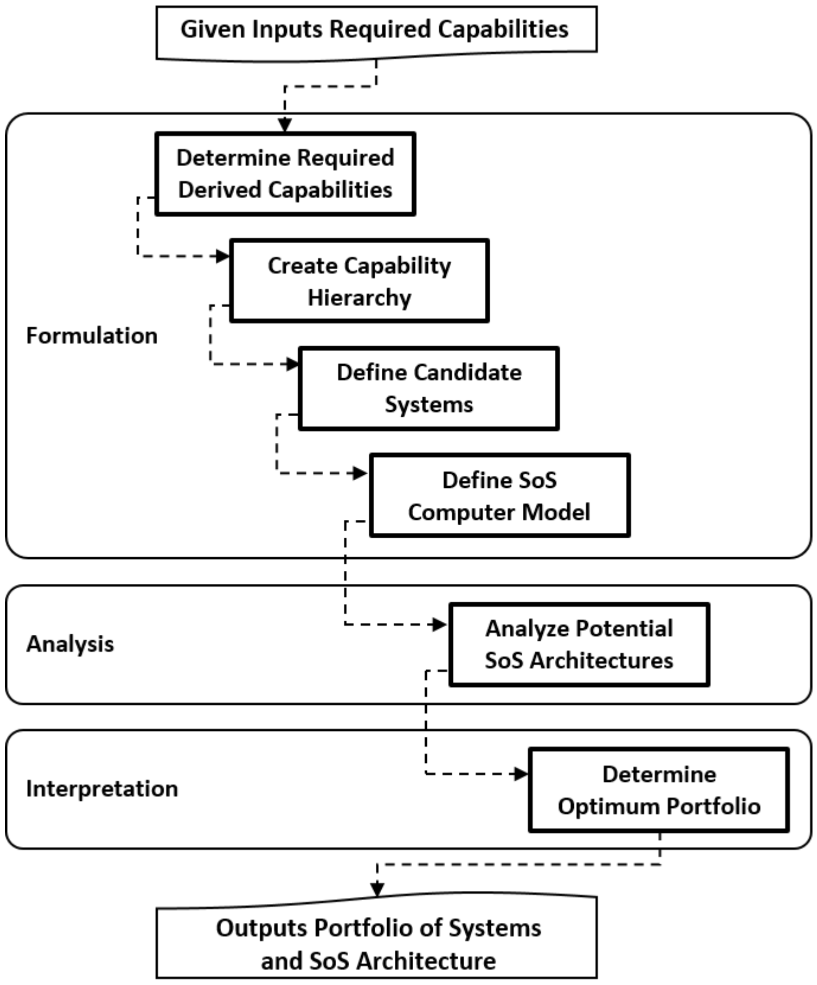

Given Inputs Required Capabilities
Formulation
Determine Required Derived Capabilities
Create Capability Hierarchy
Define Candidate Systems
Define SoS Computer Model
Analysis
Analyze Potential SoS Architectures
Interpretation
Determine Optimum Portfolio
Outputs Portfolio of Systems and SoS Architecture

**Figure 11:** The process depicted is a reproduction of the original developed by Iacobucci [50]. It formalizes the process, inputs, and outputs of the RAAM methodology. RAAM provides a method of performing analysis of large system of systems in a lightweight, memory-efficient environment. The result is the capability to perform analysis on large numbers of architecture alternatives in a short period of time.

method's process.

STASE utilizes a technique called Set Theory-influenced System Decomposition (STSD) to decompose the problem into three primary spaces: the architecture space, the design space, and the objective space [104]. A morphological approach is taken for the decomposition of the system of systems, resulting in the architecture space. The design space consists of all of the design parameters that define an architecture. This includes parameters defining both physical and functional elements. The objective

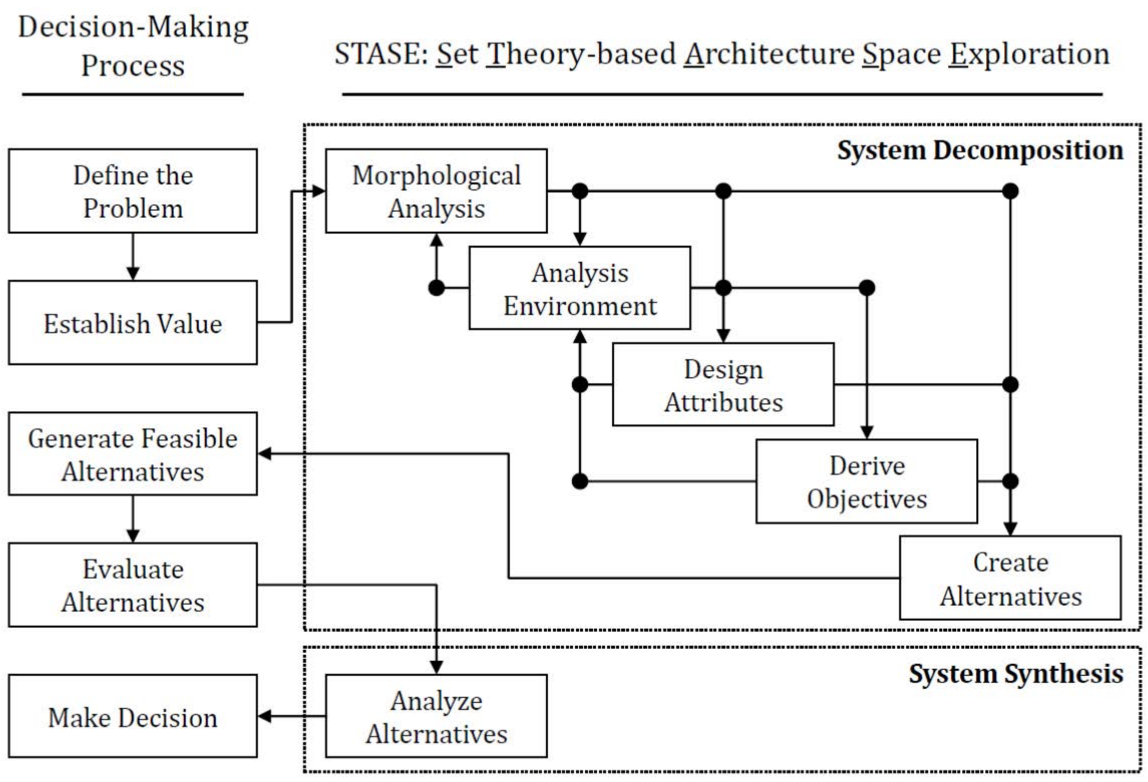


**Figure 12:** STASE attempts to reduce the challenges associated with large design spaces for system of systems by defining architectures in a novel way by utilizing set theory. The architecture is decomposed into 3 spaces which are used to translate the physical architectures into defined objectives. The intersection of the objective subsets are analyzed using Pareto analysis to determine optimum designs from the bounds of these subsets [104].

space consists of parameters which define the desired outcome of an architecture. Once the problem has been decomposed into the various spaces, alternatives are generated. These alternatives are derived from the morphological architecture space. Analysis of the alternatives does not follow the typical full factorial approach used when a system of systems is decomposed in a discrete manner. STASE utilizes set theory in an attempt to reduce the number of alternatives that are analyzed.

STASE provides the level of definition and decomposition of architectures desired for this dissertation. The method allows very large spaces to be explored efficiently against many objective metrics simultaneously. However, STASE lacks any formal definition of technologies. This lack of definition may provide a simple means of integrating technologies into the method, as opposed to reworking an existing method. A brief overview of set theory and how it is utilized in STASE in an attempt to minimize combinatorial explosion follows.

#### *2.3.2.1 Set Theory*

Set theory relies on an intersection between individual sets of designs to find the optimal design, given a set of stated objectives. Consider a notional problem where the architecture space is divided into two overlapping sets of architectures, A and B, as shown in Figure 13. If all of the architectures of these two sets are plotted on an axis consisting of two objective parameters, 1 and 2, whose values decrease with increasing improvement, a Pareto frontier can be drawn around the intersecting region. This frontier represents the optimal architectures for the stated objectives while also meeting the design criteria defined by both architecture sets. A Pareto finding algorithm can then be used to find this frontier from the interaction of the two architecture sets. By utilizing the Pareto finding algorithm, one does not need to perform analysis of every single possible architecture to find this optimum frontier. Rather, an optimizer is introduced to reduce the number of architectures that are evaluated to find the optimum design. It is this principle that STASE utilizes in an attempt to reduce the number of alternatives analyzed for large system of systems design problems.

### 2.3.3 MATE-CON

The multi-attribute tradespace exploration and conceptual design (MATE-CON) is the name given to the joining of the multi-attribute tradespace exploration (MATE) process and the integrated concurrent design (ICE) process [98, 96, 72]. MATE-CON breaks the complex system of systems design problem into two levels. The first is the architecture level, performed by MATE [98, 97]. The second is more detailed conceptual level design of specific architecture elements, enabled by ICE [72]. The

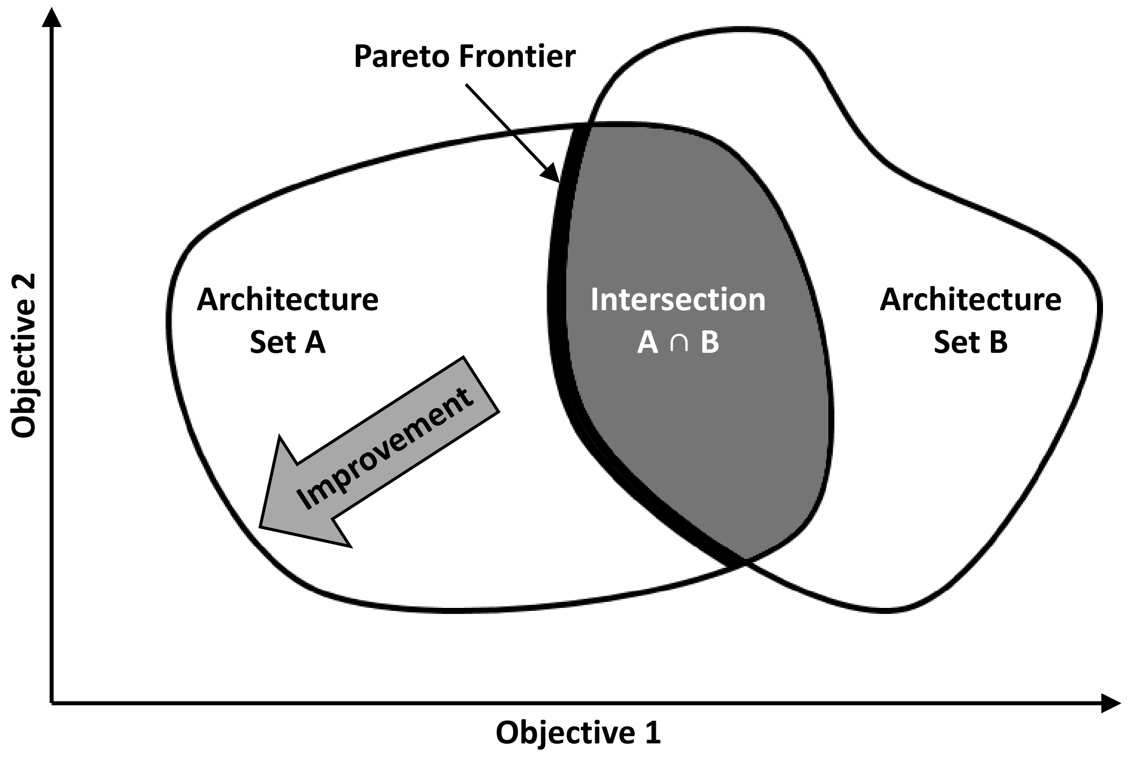


**Figure 13:** STASE utilizes set theory in an attempt to reduce the number of alternatives that must be analyzed to define the architecture space. Here, the architecture space is described by two architecture sets with an intersecting region. Translating the architecture space into the objective space, one can observe a Pareto frontier in the intersecting region, which can be found utilizing a Pareto finding algorithm.

MATE-CON process begins with an initial exploration of a large architecture space utilizing the MATE process, on the left side of Figure 14. The results of this initial MATE process feed into the ICE process on the right side of Figure 14. Here, multiple disciplines are brought together in an integrated design environment where conceptual-level vehicle designs are evaluated. Each of the disciplines are represented by some form of model operated by a human in the loop to achieve a result. This information is maintained via an electronic database which allows each of the disciplines to retrieve relevant information about other disciplines. This database of information at the conceptual vehicle design level then feeds back into the MATE process to further refine the architecture-level analysis.

The MATE process captures the high-level customer objectives and needs, which

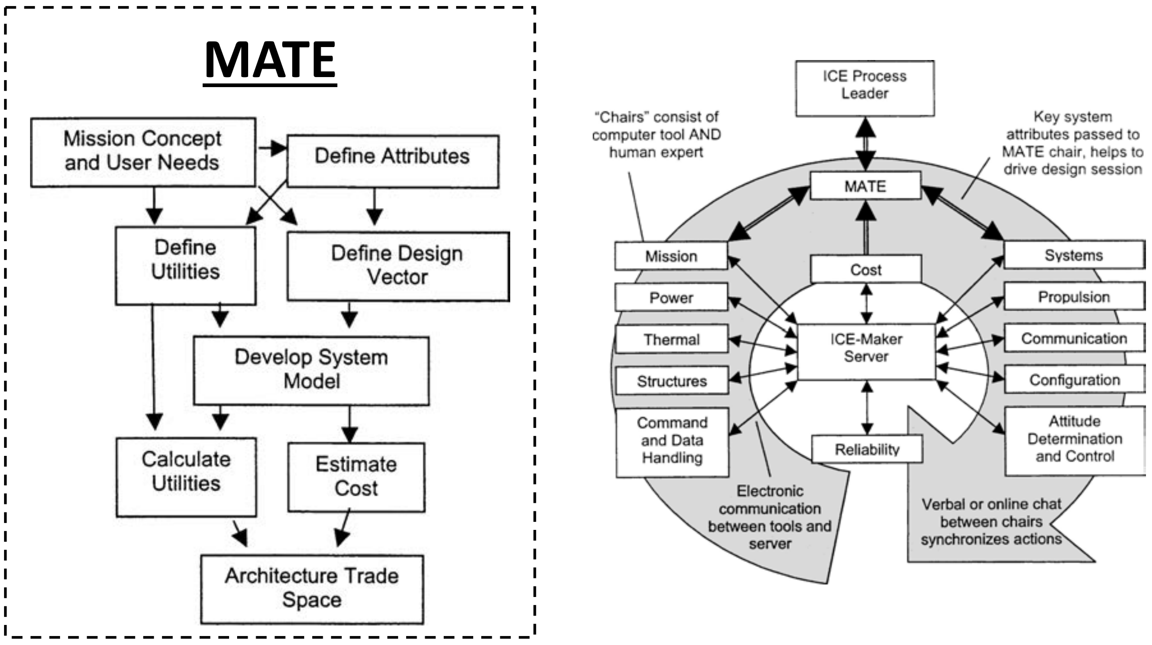

MATE
Mission Concept and User Needs
Define Attributes
Define Utilities
Define Design Vector
Develop System Model
Calculate Utilities
Estimate Cost
Architecture Trade Space
ICE Process Leader
"Chairs" consist of computer tool AND human expert
Key system attributes passed to MATE chair, helps to drive design session
MATE
Cost
ICE-Maker Server
Reliability
Mission
Power
Thermal
Structures
Command and Data Handling
Systems
Propulsion
Communication
Configuration
Attitude Determination and Control
Electronic communication between tools and server
Verbal or online chat between chairs synchronizes actions

**Figure 14:** MATE-CON unites multi-attribute tradespace exploration (MATE) and integrated concurrent engineering (ICE) to provide a capability to assess many design choices, quantitatively, very early in the design process [72].

are then mapped to specific design variables which a modeling and simulation environment operates on to calculate the figures of merit. This process populates a architecture trade space which can be explored to refine the problem to an architecture or set of architectures with which to move forward. The analysis process is highly automated, tailored to the problem in question. Results are presented in a two-dimensional multi-objective space consisting of utility and cost. Utility is a dimensionless value which represents the overall performance of a design utilizing a multi-attribute utility process to consolidate multiple figures of merit into a single utility value. Cost is simply the physical currency cost of a design.

The ICE process aims to capture more of the detailed design information of an architecture, typically present in the vehicle element. The various subsystem disciplines of the design are integrated into a human-in-the-loop physical design environment. Here, subsystems and technologies may be traded and evaluated. The goal is to ensure feasibility of a given architecture at the vehicle level. If an infeasible design results from this process, an alternative architecture from the MATE process would be selected and the ICE process restarted.

MATE-CON begins to integrate subsystem-level analysis of elements within an architecture design framework. However, the method has a disconnect between the two. The MATE process provides the high-level analysis of architectures against the customer objectives. At this level, the elements of the design are viewed only at the system level. It is not until detailed design of the vehicle element, through the ICE process, that subsystem-level analysis is introduced. Because MATE-CON performs a down selection of architecture alternatives in the MATE process before subsystem-level conceptual vehicle design via the ICE process, the interaction between subsystems and the architecture may be difficult to observe. Additionally, MATE-CON provides no formal definition of technologies. It is possible to perform technology evaluations due to the subsystem-level vehicle definition in ICE; however, due to the down selection of architectures before more detailed conceptual design of vehicle elements, observing the effects of technologies at the architecture level is challenging. Finally, MATE-CON does provide a multi-objective analysis of architectures, though it is consolidated to only two dimensions at the highest level, which may obscure further exploration of the architecture space when considering subsystem-level design choices.

## *2.4 Technology Evaluation Methods*

A quick literature search will provide numerous methods developed to perform technology evaluation. The typical goal of any technology evaluation technique is to provide a clear understanding of the technologies of interest. This will present decision makers with an understanding of what technologies may be the best choice for a potential solution to a given problem. Presented here are a few of the more relevant methods that have been developed. These methods define technology in a variety of ways, some of which differ from the definition of technologies as presented by this dissertation. However, the approaches that these methods take provide insight into what will be desirable traits for integrating such a process into a new framework.

### 2.4.1 TIES

Historically, system design in the aerospace industry held a paradigm where the primary objective was to maximize performance while minimizing weight. However, as economic and performance objectives became increasingly strict, focus shifted from a performance-based design paradigm, to one of affordability and quality [57]. To meet these strict technical and financial requirements, new technologies had to be considered as part of the solution. Technology Identification, Evaluation and Selection (TIES) is one of the initial methods developed to meet this design paradigm shift, allowing designers to analyze the impact technologies have on baseline designs from both a performance and economic perspective, while also helping decision makers with selecting technologies worth investing in. Figure 15 shows the flow of the TIES methodology.

The initial steps of TIES focus on defining the problem and design space and analyzing that design space in an attempt to understand whether infusing technology is required to solve the problem. In these initial steps, many standard engineering tools are utilized, such as quality function deployment, to aid in translating customer requirements into engineering metrics, and morphological matrices to understand the physical breakdown of the design space. Analysis is performed utilizing techniques such as Response Surface Methodology and Monte Carlo simulations. These techniques allow probabilistics and uncertainty to be included in the analysis of designs and technologies.

Once feasibility and viability have been determined, the method continues with technology identification, evaluation, and selection, as its name implies. These steps

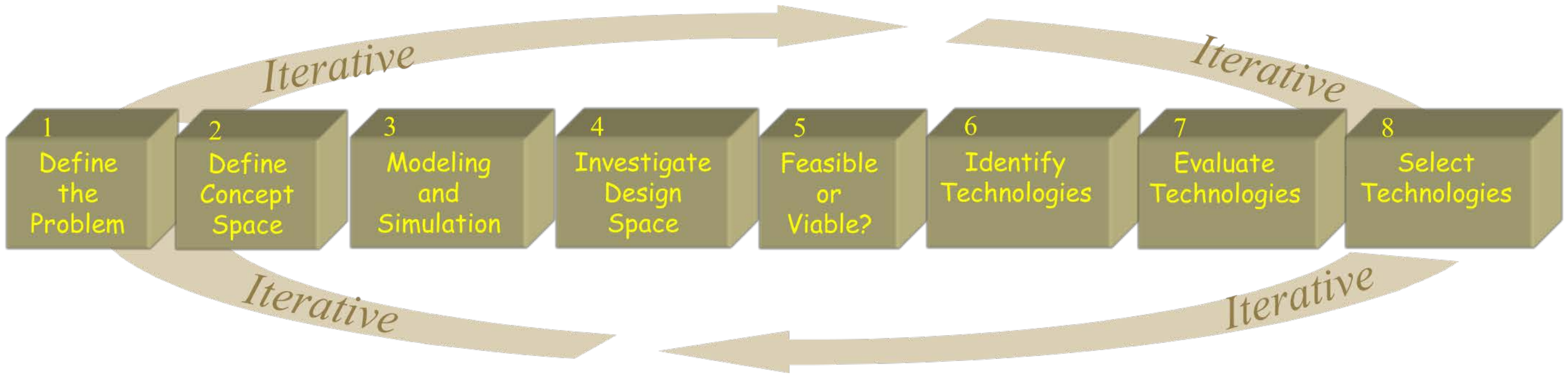


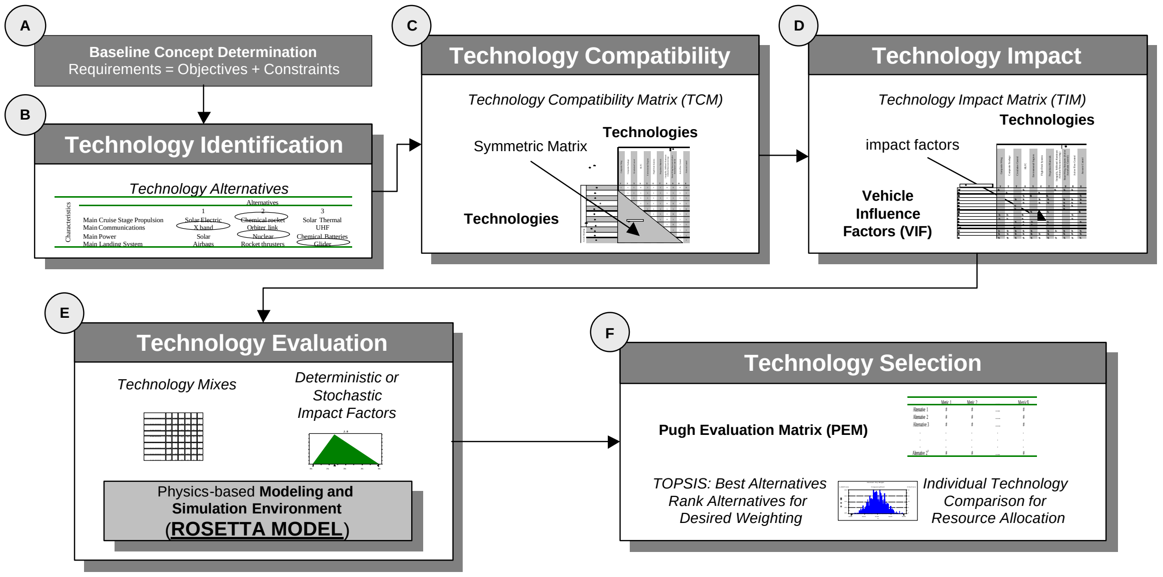
**Figure 15:** The TIES methodology was developed to meet the increasing demand on designers to perform design from a more economic approach, as opposed to the historical performance-based approach. TIES begins by defining the customer requirements and design space through standard engineering methods such as quality function deployment. Methods such as response surface equations and Monte Carlo simulation are employed to analyzing the existing design space to determine whether technology infusion is necessary. Once deemed necessary, technologies are infused into the design and their impact analyzed and quantified to reach a final decision on a technology-infused design [57].

are only performed if it is determined that technologies are required to meet the customer requirements set forth in the initial steps of the method. By considering technology compatibilities and impact mappings on specific engineering design metrics, the effects of infusing technology into a given design can be modeled and assessed, including technology impact uncertainties. TIES uses a concept called "K-factors" as adjustments to the engineering design metrics as a method of inserting technology impacts into the already existing modeling and simulation framework developed in the early stages of the methodology. This allows for a quick turnaround on analysis of a wide range of technologies without a large investment in developing new models. The final step of the TIES methodology consists of utilizing decision making techniques, such as Multi-Attribute Decision Making (MADM), to aid decision makers in making final choices on a technology family.

TIES provides a well-formulated and robust means of evaluating technologies at a subsystem level. It allows many technologies or sets of technologies to be evaluated side by side, considering a variety of objectives simultaneously. Additional, TIES allows statistical evaluation of technology performance and risk. The method was primarily developed for the evaluation of technologies on a specific baseline design. Additional, designs are not defined at the architecture level as defined by this body of work.

#### *2.4.1.1 ATIES*

Abbreviated Technology Identification, Evaluation, and Selection (ATIES) is a method developed by A.C. Charania as subset of the TIES methodology to suit the space transportation industry [17]. ATIES removes the initial steps of TIES relating to characterizing the mission needs and need for technologies on the basis that these are generally known to the designer in the space transportation conceptual design community. Furthermore, ATIES omits the technology identification step initially intended to identify available technologies for infusion. Again, it was stated that these technologies are well known to the designer as a requirement for the design [87].



A
Baseline Concept Determination
Requirements = Objectives + Constraints
B
Technology Identification
Technology Alternatives
C
Technology Compatibility
Technology Compatibility Matrix (TCM)
Symmetric Matrix
Technologies
Technologies
D
Technology Impact
Technology Impact Matrix (TIM)
Technologies
impact factors
Vehicle Influence Factors (VIF)
E
Technology Evaluation
Technology Mixes
Deterministic or Stochastic Impact Factors
Physics-based Modeling and Simulation Environment
(ROSETTA MODEL)
F
Technology Selection
Pugh Evaluation Matrix (PEM)
TOPSIS: Best Alternatives Rank Alternatives for Desired Weighting
Individual Technology Comparison for Resource Allocation

**Figure 16:** The Abbreviated Technology Identification, Evaluation, and Selection (ATIES) method is a subset of the larger Technology Identification, Evaluation, and Selection (TIES) method tailored to the space transportation community. Steps pertaining to problem, design space, and technology definition are omitted, while maintaining the core identification, evaluation, and selection steps [17, 87].

ATIES maintains steps related to technology compatibility and impact matrices, modeling and simulation frameworks utilizing methods such as response surface equations and Monte Carlo simulation, and final selection through some means of decision making, typically a weighted combination of figures of merit. ATIES makes an addition to the TIES method by including a filter which allows technology families that do not fit within a defined budget to be automatically eliminated in an attempt to manage the large combination of architectures which exist in space transportation design problems. A general flow diagram of the ATIES method is provided in Figure 16. Because ATIES was developed as a subset of the TIES method, it inherits the advantages and disadvantages with regard to desired features for a method or framework to meet the statement of purpose of this dissertation.

### 2.4.2 TRIPS

Many of the technology evaluation methods currently in use rely on a technology's impact on underlying subsystem-level metrics, such as an engine's thrust or specific impulse, to provide an overall effect on the subsystem and related systems and architecture. These "K-factors" provide a simple method for applying technology impacts. However, many of the methods stop at this point, providing little information about the development of a technology and its impact on a design. The Technology Roadmapping and Investment Planning System (TRIPS) attempts to solve this problem by modeling a technology's development in a probabilistic manner and then generating an optimal resource allocation profile given a set of programmatic constraints [18]. The goal is to bring information related to the cost, schedule, and uncertainty of technology development into the design of architectures to supplement the performance impacts estimated by other methods.

TRIPS models the development of a technology as a discrete time Markov Chain. Each event is a transition from one TRL to the next. The probability that a technology will transition to the next TRL in a single time step of the model is a function of the monetary investment in that technology. This transition probability is defined using a statistical distribution, such as a triangular or normal distribution of the likelihood of transition versus the estimated cost. A matrix for each technology is generated, representing all the probabilities for the developments of a technology through the nine TRLs. These transition probability matrices populate the probability catalog. Figure 17 shows the process by which portfolios of technologies are evaluated, given user-defined funding profiles and architectures. This information is analyzed to determine feasible portfolios for maturing a desired capability [18].

TRIPS includes the formal definition of architectures within the method. Additionally, the evaluation of multiple technologies against many objectives is performed.

However, the level of definition of architectures is not at the subsystem level as required. The method focuses on high-level architecture objectives to measure the desirability of technologies as well, not meeting the needs for an integrated architecture design and technology evaluation framework.

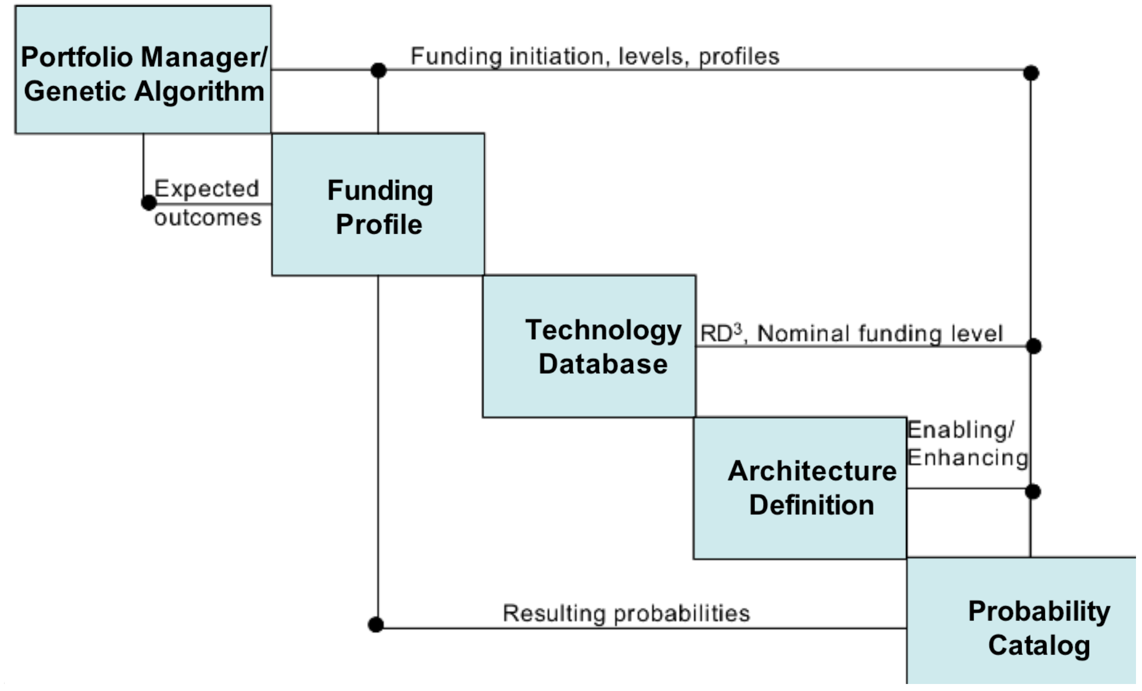


**Figure 17:** The Technology Roadmapping and Investment Planning System (TRIPS) is focused on solving the problem of technology development planning. It approaches technology development in a probabilistic manner through a discrete Markov Chain simulation to account for variations in investment funding levels and their effect on technology development and associated architecture capabilities [18].

### 2.4.3 TAPP

The Technology Alignment and Portfolio Prioritization (TAPP) technique attempts to define and assess sets or portfolios of desired technologies while incorporating organizational structure, available resources, and policies into the analysis. Funaro recognized the challenges associated with assessing technology for large-scale systems of systems, claiming that the time-intensive nature of technology evaluation lends itself well to automation [32]. Because of this, TAPP approaches the problem by looking at how technologies align with an organization and its desired missions, in this case, NASA's Marshall Space Flight Center. The quantitative and fiscal impact on technology analysis are left to other evaluation methods such as TIES and TRIPS as discussed in sections 2.4.1 and 2.4.2, respectively. Figure 18 shows the basic process of TAPP.

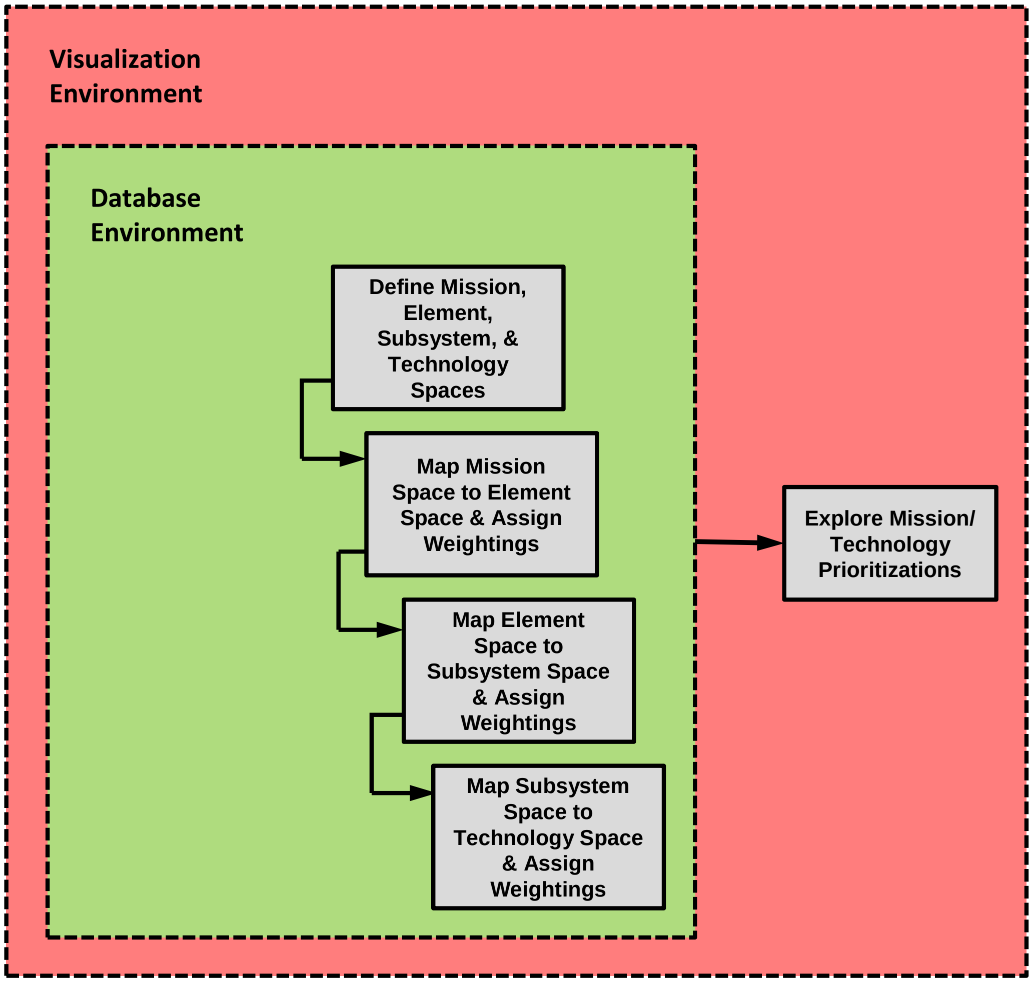

Visualization Environment
Database Environment
Define Mission, Element, Subsystem, & Technology Spaces
Map Mission Space to Element Space & Assign Weightings
Map Element Space to Subsystem Space & Assign Weightings
Map Subsystem Space to Technology Space & Assign Weightings
Explore Mission/ Technology Prioritizations

**Figure 18:** The Technology Alignment and Portfolio Prioritization (TAPP) is a method that focuses on determining how well a portfolio of technologies aligns with a defined organization's competencies and goals [32]

TAPP requires the definition of four spaces: mission, element, subsystem, and technology. The mission space is the desired set of missions the organization is interested in performing. The element space is all of the physical systems and systems of systems that are available to an organization. The subsystem space is the set of subsystems required to define all elements in the element space. The technology space is the set of all considered technologies available to the organization. During technology definitions, it is also important to define how the organization's capabilities align with each technology. Once these four spaces are defined, mappings must be made to link the mission space to the technology space, through the elements and subsystems spaces. To do this, there are three mappings connecting the four spaces:

$$\textbf{Mission}\longrightarrow\textbf{Element}\longrightarrow\textbf{Subsystem}\longrightarrow\textbf{Technology}$$

Each mapping has its own set of user-defined weighting parameters. Once all of the spaces, mappings, and weighting parameters are defined, the technology prioritization can be explored based on the desired mission. Because an alignment with each technology was made with the organization's capabilities, not only can users explore how technologies are prioritized for each mission, but also how well the technology portfolio, and hence associated missions, align with the organization's capabilities. This can provide valuable information regarding gaps in an organization's workforce.

TAPP inherently defines technologies at the subsystem level of the design. However, the effect of the technology on a physical design is not accounted for. Though multiple technologies may be evaluated side by side, they are only evaluated on a single objective, organizational alignment. This does not meet the needs of a framework for this research. Additionally, the method does not consider architectures as defined by this document. Rather, it traces vehicle needs through a specific mission.

### 2.4.4 QuantUM$^3$

The Quantitative Uncertainty Modeling, Management, and Mitigation (QuantUM$^3$) method was developed to allow designers the ability to understand the risk associated with different technologies in an architecture by leveraging uncertainty quantification techniques [36]. QuantUM$^3$ integrates cost and schedule into the analysis of technologies through quantitative, probabilistic performance analysis of the risk associated with the readiness and effectiveness of each technology in a portfolio [36]. The flow of the QuantUM$^3$ methodology is shown in Figure 19.

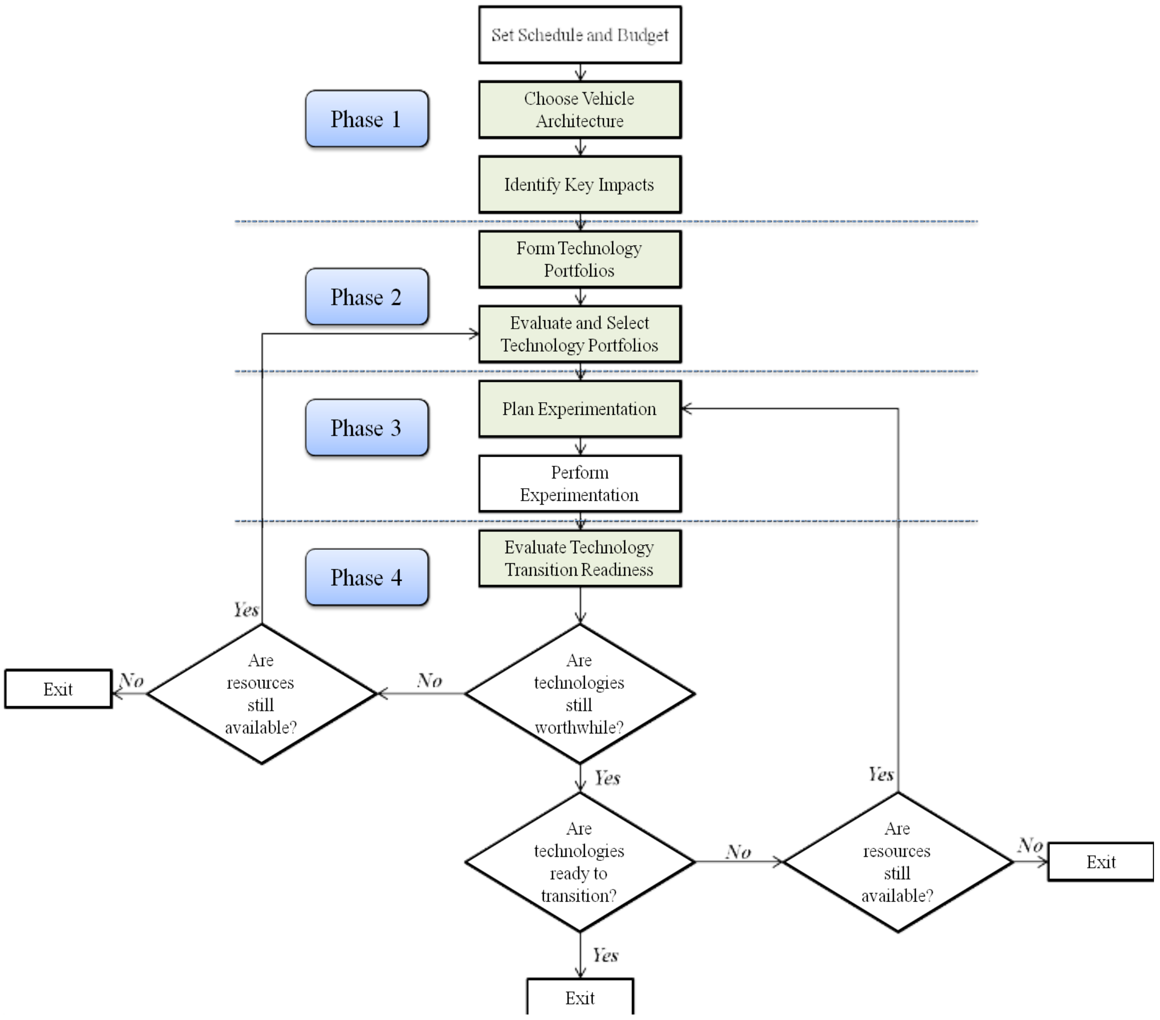

Set Schedule and Budget
Phase 1
Choose Vehicle Architecture
Identify Key Impacts
Form Technology Portfolios
Phase 2
Evaluate and Select Technology Portfolios
Phase 3
Plan Experimentation
Perform Experimentation
Phase 4
Evaluate Technology Transition Readiness
Yes
Exit
No
Are resources still available?
No
Are technologies still worthwhile?
Yes
Yes
Are technologies ready to transition?
No
Are resources still available?
No
Exit
Yes
Exit

**Figure 19:** The Quantitative Uncertainty Modeling, Management, and Mitigation (QuantUM$^3$) method focuses on the risk associated with technologies in an architecture through uncertainty quantification techniques [36].

As is seen in the flow diagram, a baseline architecture is selected in phase 1. This allows technologies to be analyzed from a risk-based performance assessment, while also including the effects of cost and schedule. This provides a very detailed analysis of various technologies for the defined architecture and organizational constraints. Although QuantUM$^3$ allows the evaluation of multiple technologies on a multi-objective basis, it fails to define architectures and technologies at the proper level. Additionally, the methods focuses on technology evaluation after architecture down selection.

## 2.5 Model-Based Systems Engineering

Literature provides many formal and informal definitions for the terms Model-Based Systems Engineering (MBSE). INCOSE provides a formal definition for Model-Based Systems Engineering as: "...the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases"[48]. Here we see MBSE as a way of formalizing the application of modeling to support systems engineering tasks. However, the definition provided by The National Defense Industrial Association helps to clarify by defining MBSE as "An approach to engineering that uses models as an integral part of the technical baseline that includes the requirements, analysis, design, implementation, and verification of a capability, system, and/or product throughout the acquisition life cycle"[9]. In this definition, it is made much more clear that modeling should be an integral part of the systems engineering process. Both definitions are clear and specific that this integration should not just occur in the analysis phase of a design, but rather throughout the entire life cycle, from requirements formulation, conceptual design, and through the end of life phases.

The diagram in Figure 20 shows a high-level overview of the MBSE ontology. The ontology is broken into seven high-level concepts [49]:

- **System** concepts, which cover the basic concepts associated with System, Systems of Systems, Constituent Systems, etc.

- **Need** concepts, which cover all concepts associated with System Needs, such

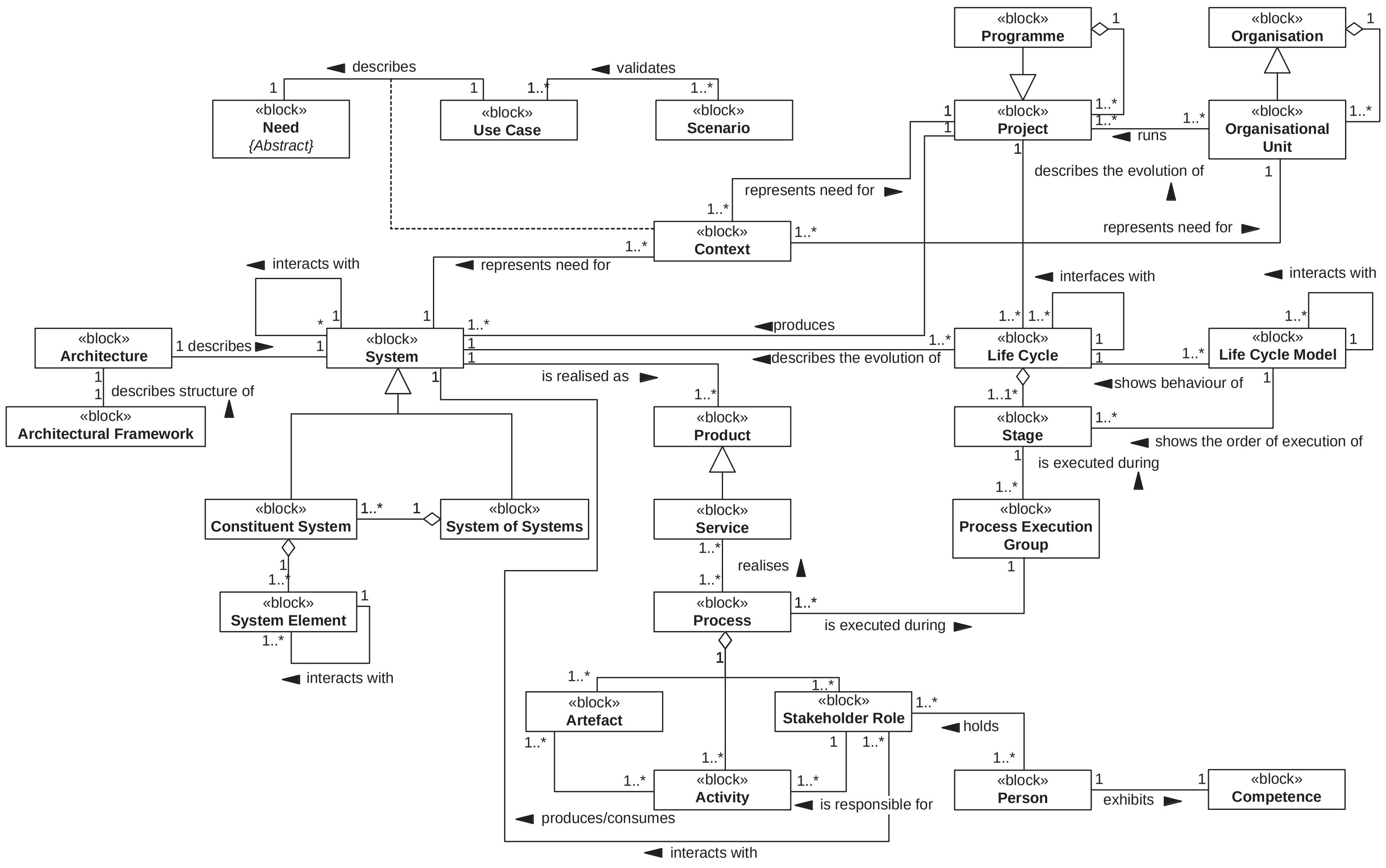


**Figure 20:** The high-level Model-Based Systems Engineering Ontology[49]

as Requirements, Capabilities and Goals.

- **Architecture** concepts, where the description and structure of Architectures using Architectural Frameworks are discussed.
- **Life Cycle** concepts, where different Life Cycles and Life Cycle Models are discussed, along with the interactions between Life Cycles.
- **Process** concepts, where the structure, execution and responsibility of Processes are discussed.
- **Competence** concepts, where the ability of people associated with Stakeholder Roles are defined.
- **Project** concepts, where Project and Program-related concepts are defined.

These seven high-level concept groups can be seen graphically in Figure 20, as well as their relation to each other. MBSE shifts from a traditional document-based record of authority to a more model-centric data-rich environment. This allows teams and engineers to more quickly and readily understand design changes and how they impact a design [42]. The overarching objective of MBSE is to provide more systems engineering depth during design while also reducing acquisition time without increasing cost. However, MBSE is not without its drawbacks. Initializing a model-based engineering approach requires investment in tools, training, and infrastructure that is not required by a document-centric system. Also, MBSE does not replace the need for rigorous, detailed disciplinary design teams [9]. Furthermore, the highly structured nature of MBSE may provide additional challenges during the early conceptual design phases where many fundamentally different designs are being considered side by side.

Beginning in the late 20th century, modeling languages began to be created to formalize how models were developed and integrated within industry. These languages, similar to programing languages, provided formal, object-oriented frameworks which allowed concepts such as MBSE to be implemented in practice. The remainder of this section is dedicated to providing a high-level overview of three of these languages: the Unified Modeling Language (UML), the Systems Engineering Modeling Language (SysML), and the Architecture Analysis Design Language (AADL).

### 2.5.1 The Unified Modeling Language

UML was developed initially by Grady Booch, Ivar Jacobson, and James Rumgaugh as a result of the Object Management Group's (OMG) calling for specification of a uniform modeling standard in 1996 [103]. UML represents models in the form of diagrams, which provide a view of a specific part of reality described by a model. They are a form of functional decomposition of a system. Figure 21 shows the taxonomy of the various types of diagrams defined in UML.

Through these various diagram types, systems and systems of systems may be presented in various ways which help to describe them in a way which is consistent among different teams of people. This allows for communication between different groups without having to worry as much about language usage and other model miscommunication, allowing people to focus more on the design. For a detailed description of the diagrams presented in Figure 21, the reader is referred to Chapter 2, Section 3 of UML@Classroom: An Introduction to Object-Oriented Modeling, by Seidl et al [103].

Though UML provided the first steps in providing a general framework for defining and presenting systems which could help communication among designers, it is not without its flaws. Because UML was developed with the intent of widespread use among many disciplines, it was intentionally left vague in its description of its various components such that it could be molded to the user's specific needs. However, it is this generality which may give rise to miscommunication as different teams interpret

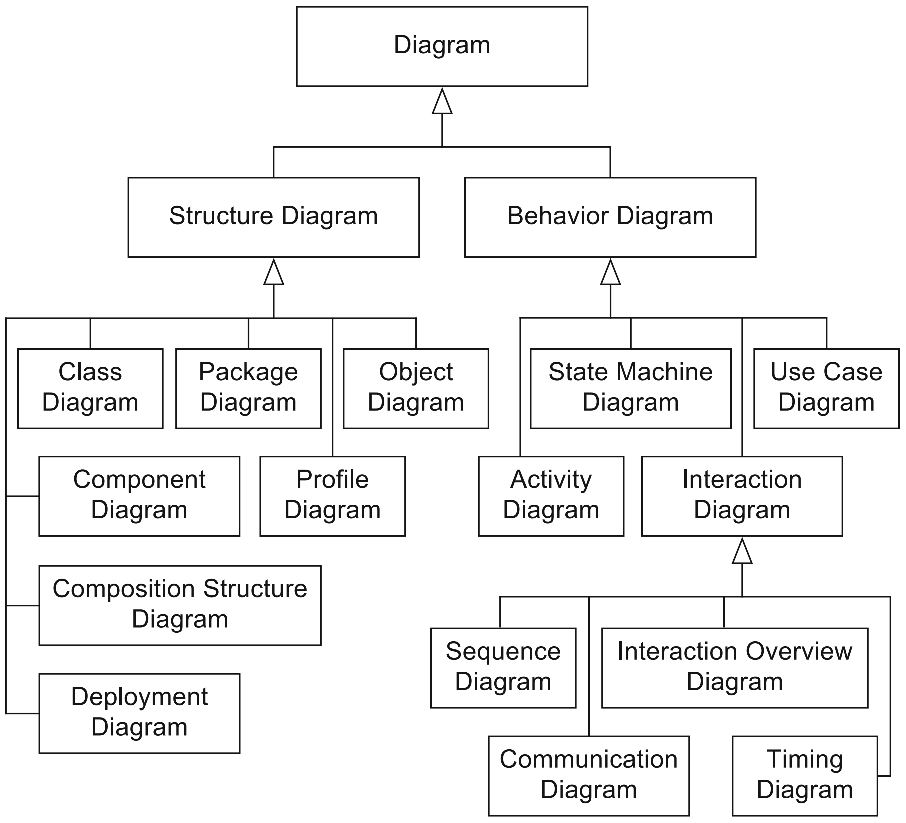


**Figure 21:** Unified Modeling Language Diagram Taxonomy[103]

the vague guidelines differently. Due to this observation, in 2003 the OMG issued the "UML for Systems Engineering Request for Proposal" following a decision by INCOSE to customize UML for systems engineering applications [49, 43].

### 2.5.2 The Systems Engineering Modeling Language

As a result of the OMG's call for proposals, a broad base of tool vendors, industry users, government agencies, and professional organizations worked to develop standards to meet the request. In 2006, the OMG announced the adoption of the SysML specification and subsequently published v1.0 in 2007 [43]. Up through mid-2017, the OMG has published a total of 5 revisions to the original specification, the most recent being published in May of 2017 [86].

Because SysML was developed as a result of tailoring UML to better suit systems engineering problems, it is only natural that these two design languages share a commonality. Figure 21 is a graphical depiction of the relationship between SysML and UML. As can be seen, SysML makes use of a large portion of UML. Overall, SysML has a smaller footprint than UML. In all, SysML defines only 9 diagrams, compared to UML defining 13. Some parts of UML were considered unnecessary for the systems engineering domain and were left out of the new language. In addition, SysML adds a few new diagrams and constructs not present in UML to better define systems engineering problems. Also, those parts of UML used by SysML were subject to changes to better suit systems engineering paradigms. Figure 23 shows the relationship between the SysML diagrams, as well as the relations they have to UML diagrams. For details regarding the various diagrams used in SysML, refer to SysML for Systems Engineering by Jon Hold and Simon Perry [49].

### 2.5.3 The Architecture Analysis and Design Language

The Architecture Analysis and Design Language (AADL) was developed independently from UML and SysML. In November of 2004, the Society of Automotive Engineering (SAE) released the aerospace standard AS5506, which defines AADL [28]. In his introduction to AADL, Peter Feiler states that AADL is "...a textual and graphical language used to design and analyze the software and hardware architecture of real-time systems and their performance-critical characteristics."[29] AADL was developed to support the avionics, aerospace, and automotive industries, used to describe the structure of such systems as an assembly of software components.

Unlike UML and SysML, the AADL is not as graphical, but rather is written more like a programming language to describe components and their relationships. The AADL defines three distinct sets of component categories:

1. **Application Software**

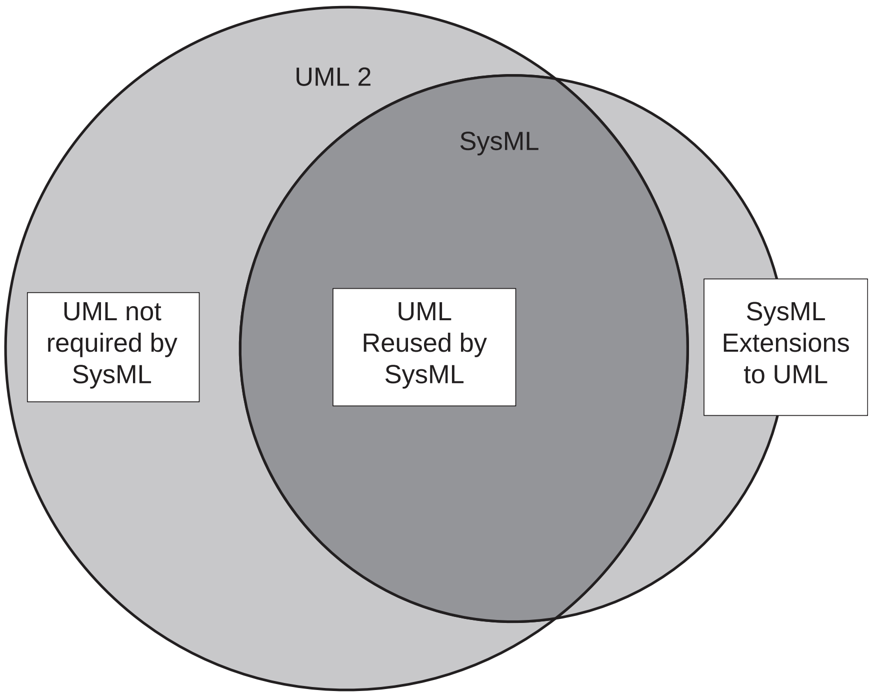


**Figure 22:** SysML and UML share a common root of diagrams. However, SysML omits diagrams less relevant to system engineering problems, while also extending the base functionality of UML to improve modeling of systems engineering problems [49, 43].

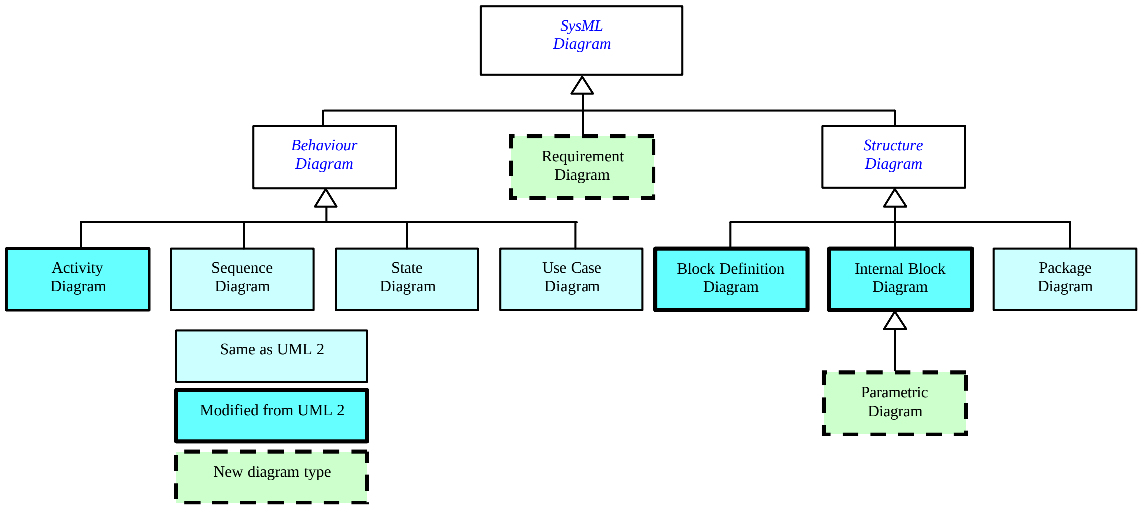


**Figure 23:** Systems Engineering Modeling Language Diagram Taxonomy [31, 43]. As can be seen, SysML and UML share a common set of diagrams. However, SysML makes modifications to a few diagrams, while also add a few new diagrams, making the language more suited for the systems engineering domain.

   (a) **Thread:** a schedulable unit of concurrent execution

   (b) **Thread Group:** a compositional unit for organizing threads

   (c) **Process:** a protected address space

   (d) **Data:** data types and static data in source text

   (e) **Subgroup:** callable sequentially executable code

2. **Execution Platform**

   (a) **Processor:** components that execute threads

   (b) **Memory:** components that store data and code

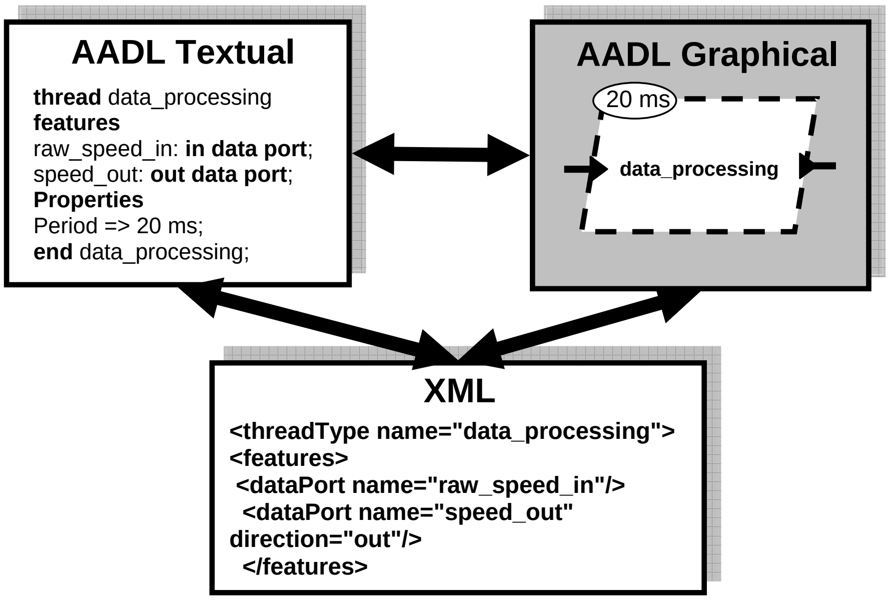
   (c) **Device:** components that interface with and represent the external environment

   (d) **Bus:** components that provide access among execution platform components

3. **Composite**

   (a) **System:** a composite of software, execution platform, or system components

Components have type declarations and implementations, which help to define a component's externally visible characteristics and its internal structure, respectively [28]. These components are brought together to form a full representation in the AADL. The model can be expressed textually, graphically, or as an extensible markup language (XML) document. Figure 24 summarizes the alternative representations, showing samples of each. For further details regarding AADL and components, refer to The Architecture Analysis & Design Language (AADL): An Introduction by Feiler et al. [28].



```
thread data_processing
features
raw_speed_in: in data port;
speed_out: out data port;
Properties
Period => 20 ms;
end data_processing;
```

```
<threadType name="data_processing">
<features>
 <dataPort name="raw_speed_in"/>
  <dataPort name="speed_out"
direction="out"/>
  </features>
```

**Figure 24:** Notional Architecture Analysis & Design Language Model Representations [28]

# CHAPTER III

# FRAMEWORK FORMULATION

This chapter will explore the gaps in current methods in meeting the needs of an architecture evaluation method which fulfills the statement of purpose of this research. These gaps will form the primary research objective of this dissertation, as well as the driving needs of a new method which will be developed through a discussion of topics relevant to such a methodology. Throughout this chapter, research questions and hypotheses will be developed, forming the structure of this research.

## *3.1 Gaps In Current Methods*

Based on the definitions for architectures and technologies, combined with the general statement of purpose of this dissertation, a set of required features for evaluating methods and techniques may be developed. The statement of purpose for this dissertation is reprinted below for reference.

**Statement of Purpose**

To provide a capability to analyze complex systems of systems to an extent which will provide decision makers in the early phases of design sufficient information to reduce the risks associated with cost and schedule overruns due to lack of design knowledge.

In Chapter 2, a collection of methods and frameworks was discussed, along with potential advantages and disadvantages with regard to the desirable features of a method or framework capable of meeting the needs posed by the statement of purpose of this dissertation. This information is summarized in Table 5, showing each method's ability to meet the required features list. It becomes self-evident that no one method meets all the feature requirements identified by this research. The deficiencies among these methods and techniques in meeting the primary statement of purpose of this research highlights the important characteristics desirable for a new method to be developed. The following gaps have been identified within the set of methods described, outlining a set of desired characteristics for a new framework to integrate architecture analysis and technology evaluation at a subsystem level. Though none of the identified methods and techniques fully meet the feature requirements outlined in this section, components from them may provide a good starting point for defining a new framework which meets the required features. This will be discussed in further detail throughout the remainder of this chapter.

**Table 5:** Methods Comparison

| | | Methods | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | ARCHITECT | STASE | MATE-CON | TIES | ATIES | TRIPS | TAPP | QuantUM$^3$ |
| Features | Architectures Defined at Subsystem Level | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | — | ✗ |
| | Can Evaluate Multiple Architectures | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | — | ✗ |
| | Multi-objective Architecture Analysis | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | — | ✗ |
| | Technologies Defined at Subsystem Level | ✗ | — | — | ✓ | ✓ | ✗ | ✓ | ✗ |
| | Can Evaluate Multiple Technologies | ✓ | — | — | ✓ | ✓ | ✓ | ✓ | ✓ |
| | Multi-objective Technology Analysis | ✓ | — | — | ✓ | ✓ | ✓ | ✗ | ✓ |

**✓ = Met** **✗ = Unmet** **— = Undefined**

1. In the space transportation domain, missions are typically selected well before physical systems and technologies are developed. The architecture and technology evaluation methods presented propose selecting a baseline design in the initial steps.

2. In the space transportation domain, architectures typically consist of many systems concurrently being designed. This produces a very large architecture alternative space. Due to the discrete nature of defining a large system of systems, architecture analysis typically focuses on analyzing the full set of design alternatives. This is impractical for the large number of alternatives that exist.

3. Typically, architecture design methods which incorporate technologies define them as an entire system, and do not fit the definition of a technology as provided in this body of work. Technology evaluation methods which define technologies in a similar manner to this research typically consider a fixed architecture when evaluating technologies.

4. Technology evaluation methods tend to focus on a single aspect of how technologies affect a design.

## *3.2 Research Objective*

Section 1.2.3 outlines the primary challenges associated with the design space exploration of complex architectures. These challenges were primarily attributed to a lack of design knowledge in the early phases of design due to technical and political uncertainty. The level of design freedom decreases rapidly during the early phases of design, while committed cost rapidly increases, as shown in Figure 6. The act of bringing design knowledge earlier into the design phases will help to reduce cost and schedule overruns associated with a lack of design knowledge. However, in an effort to incorporate this design knowledge into earlier phases of design, the system of systems spaces that must be explored become impractically large to analyze. Section 3.1 outlined gaps in the current methods for performing architecture design and technology evaluation under the definitions provided. Refining the overarching statement of purpose based on these gaps leads to the primary research objective of this dissertation:

**Research Objective**

To integrate architecture analysis and technology evaluation at the subsystem level to provide a quantitative framework in an effort to increase design knowledge early in the design process.

## *3.3 General Concept Exploration Framework*

Many models exist in literature for the purpose of concept exploration and refinement. Each are unique and vary in details, but as defined by the United States Air Force, they follow three major steps [113]:

- Trade space characterization
- Candidate solution sets characterization
- Analysis

Trade space characterization requires converting needs into quantifiable boundaries while collecting potential solution ideas. Characterizing the candidate solution sets consists of refining the trade space boundaries into realizable concepts that may represent solutions to the problem. The analysis step requires evaluating the candidate solutions on a common basis such that trends can be observed and a decision made. One such model is the Georgia Institute of Technology's Integrated Product and Process Development (IPPD) model, as seen in Figure 25.

Under this model, trade space characterization consists of establishing a need, typically done through quality engineering methods to gather customer needs and requirements, which are then converted to a set of functional requirements. These drive the definition of the problem followed by a decomposition of the trade space and an establishment of the value metrics and figures of merit. Characterizing candidate

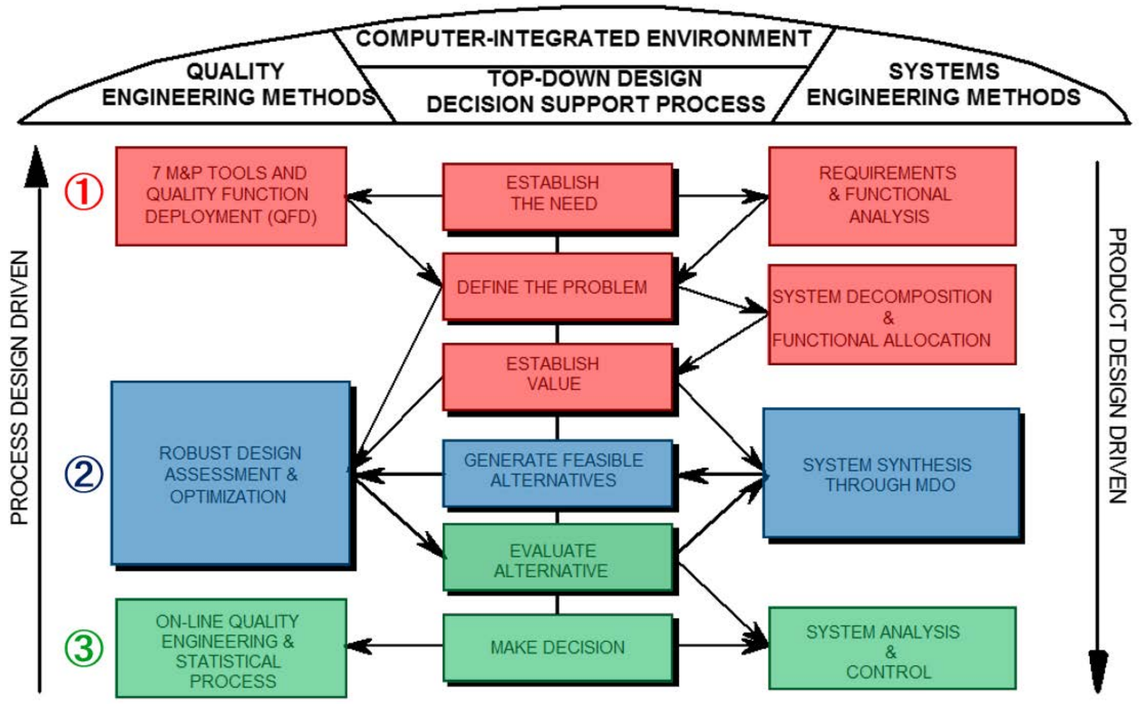
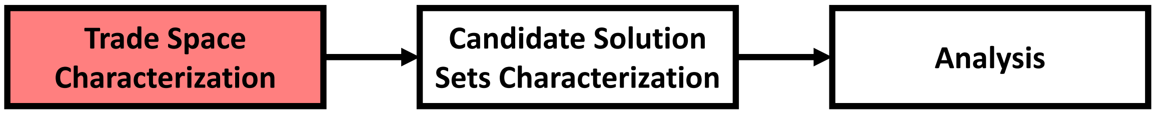

COMPUTER-INTEGRATED ENVIRONMENT
QUALITY ENGINEERING METHODS
TOP-DOWN DESIGN DECISION SUPPORT PROCESS
SYSTEMS ENGINEERING METHODS
PROCESS DESIGN DRIVEN
PRODUCT DESIGN DRIVEN
①
7 M&P TOOLS AND QUALITY FUNCTION DEPLOYMENT (QFD)
ESTABLISH THE NEED
REQUIREMENTS & FUNCTIONAL ANALYSIS
DEFINE THE PROBLEM
SYSTEM DECOMPOSITION & FUNCTIONAL ALLOCATION
ESTABLISH VALUE
②
ROBUST DESIGN ASSESSMENT & OPTIMIZATION
GENERATE FEASIBLE ALTERNATIVES
SYSTEM SYNTHESIS THROUGH MDO
EVALUATE ALTERNATIVE
③
ON-LINE QUALITY ENGINEERING & STATISTICAL PROCESS
MAKE DECISION
SYSTEM ANALYSIS & CONTROL

**Figure 25:** The Georgia Tech Integrated Product and Process Development model[101] mapped to the three major steps of a concept exploration framework as defined by the United States Air Force[113]: (1) Trade Space Characterization, (2) Candidate Solution Sets Characterization, (3) Analysis

solution sets consists of generating feasible alternatives in the trade space. Finally, analysis comprises evaluating the alternatives through system analysis and making a final decision by means of a multi-criteria decision making approach. There have been many methods developed to perform architecture design and analysis which fit a model similar to Georgia Tech's IPPD just described. In Section 2.3, two such methods, ARCHITECT and STASE, were discussed.

The primary research objective of this dissertation is not to develop an entirely new method for technology evaluation and architecture analysis. Rather, it is to integrate architecture analysis and technology evaluation at the subsystem level such that the effects of this integration may be studied. However, because there is no single method identified to meet the needs of this body of work, a basic framework for evaluating these architectures and technologies will need to be developed. In doing so, if literature provides suitable methods currently in existence that may serve as elements of this framework, then it will be desirable to integrate those elements into the required framework to achieve the main research objective. The remainder of this section will discuss the main processes required for concept exploration and how they relate to architecture design and technology evaluation. The application of relevant methods and concepts from Chapter 2 and their appropriateness as a framework for a new method will be highlighted. The research questions for this dissertation will be developed throughout the remainder of this chapter.

### 3.3.1 Phase I: Trade Space Characterization


Trade Space Characterization
Candidate Solution Sets Characterization
Analysis

The first of the three major steps in concept exploration consist of trade space characterization through definition and decomposition of the problem. This will provide insight into the relationships which exist within a design space. Because this research aims to integrate technology evaluation and architecture analysis into a single framework, the first logical research question is as follows:

**Research Question 1**

What is the relationship between architectures and technologies?

There exist many methods and techniques for decomposing a problem. Trade trees, morphological analysis, network theory, and product family design are a few examples of current methods. As is seen in Table 5, STASE is the only architecture design method which defines architectures in a similar manner to that which is required by a new framework. The method in which Jonathan Sharma decomposes the problem is called set theory-influenced system decomposition (STSD). Different system spaces–the architecture space, design space, and objective space–are defined and mapped to each other [104]. Though the technique was originally proposed for mapping spaces defined as mathematical sets, the general concept will hold for designs which are a general collection of options and parameters and does not require the strict definition of mathematical sets. The following is a brief summary of STSD and how it relates to this research. Details can be found in Section 3.1.2 of Sharma's dissertation on set theory-influenced architecture space exploration [104].

#### *3.3.1.1 Defining System Spaces*

The DoD Architecture Framework provides details on different ways of looking at an architecture. These viewpoints, as defined in version 2.0 of the document, consider many different ways of organizing data to facilitate understanding of complex system of systems architectures. Of them, the Operational Viewpoint (OV) and Systems Viewpoint (SV) are most relevant to this body of research. The most relevant SV when decomposing the problem is SV-1, the systems interface description. This viewpoint can be used to depict systems and subsystems of an architecture [115]. The most relevant OVs when decomposing the problem are OV-2, the operational resource flow description, and OV-5a, the operational activity decomposition tree. Whereas OV-5a focuses on the operational activities, OV-2 focuses on the operational activities in relation to locations. These two OVs are typically developed together due to the relationship between location and operation [115]. These viewpoints provide a way of organizing the missions of an architecture in a form that can easily be integrated into the physical systems decomposition described by SV-1. These viewpoints may be captured via morphological analysis of the problem, and are typically viewed at a high level in the form of a bat chart, which capture each of these viewpoints in a single image representing the architecture. An example bat chart is shown in Figure

26 of a notional Mars mission from NASA's Design Reference Architecture 5.0.

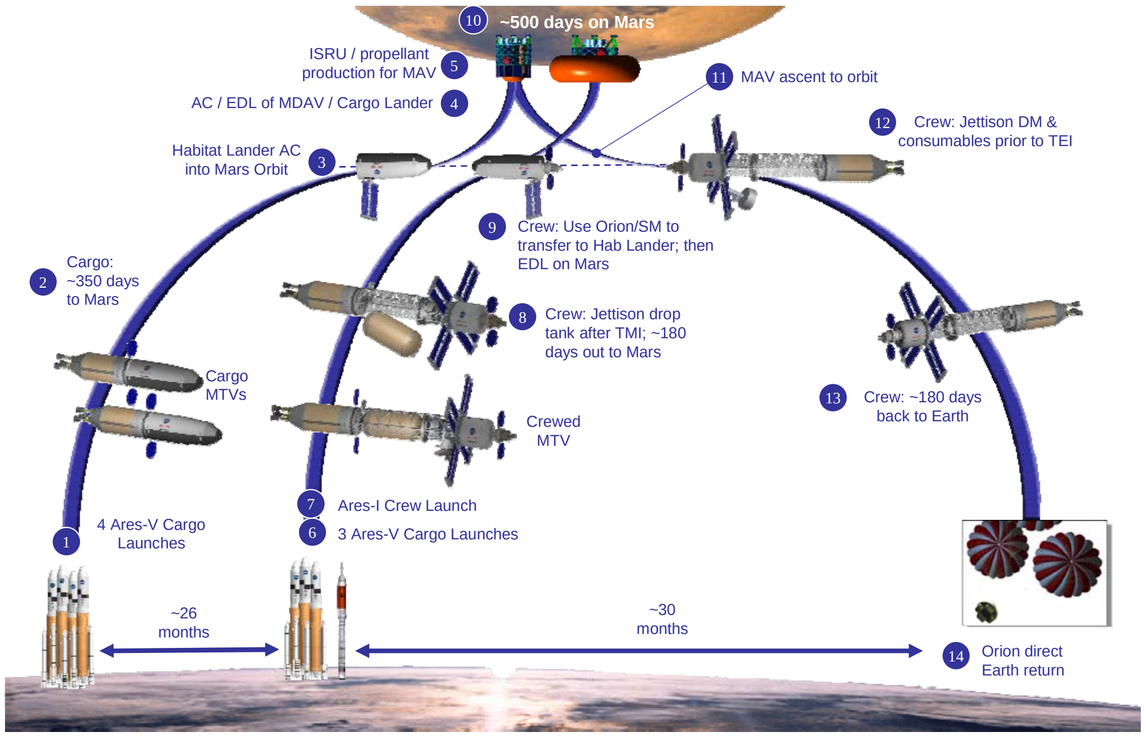
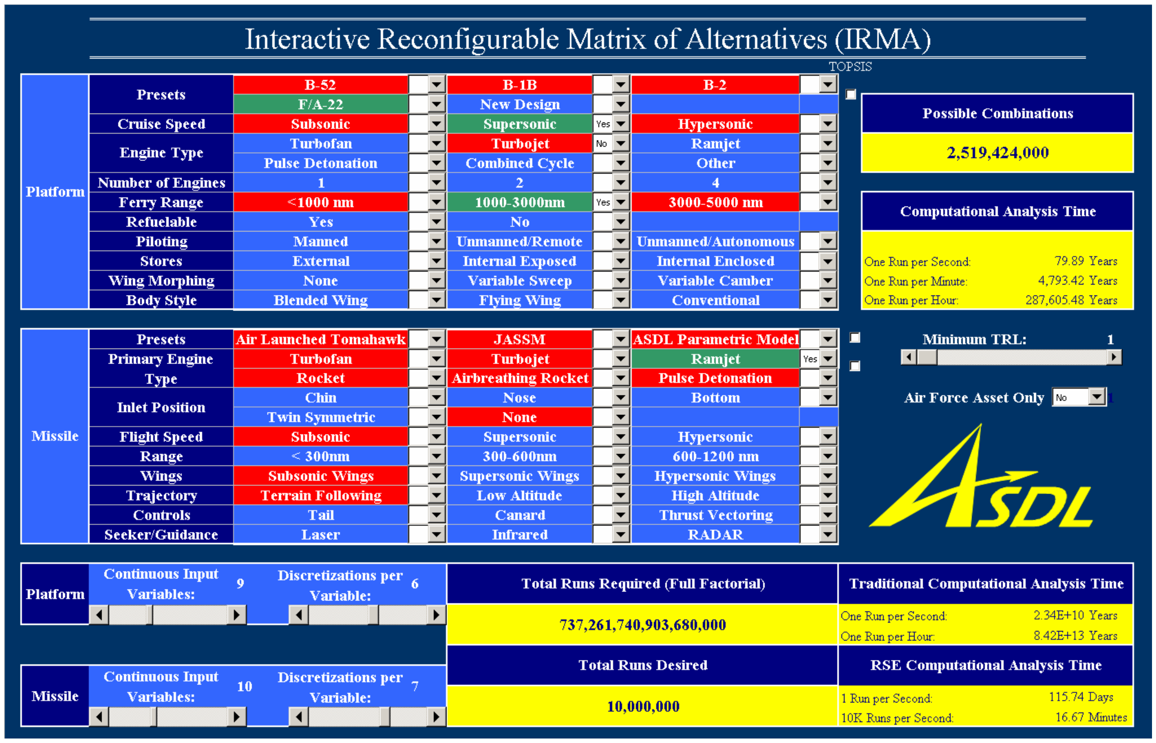

10 ~500 days on Mars
ISRU / propellant production for MAV 5
AC / EDL of MDAV / Cargo Lander 4
11 MAV ascent to orbit
12 Crew: Jettison DM & consumables prior to TEI
Habitat Lander AC into Mars Orbit 3
9 Crew: Use Orion/SM to transfer to Hab Lander; then EDL on Mars
2 Cargo: ~350 days to Mars
8 Crew: Jettison drop tank after TMI; ~180 days out to Mars
Cargo MTVs
13 Crew: ~180 days back to Earth
Crewed MTV
7 Ares-I Crew Launch
6 3 Ares-V Cargo Launches
1 4 Ares-V Cargo Launches
~26 months
~30 months
14 Orion direct Earth return

**Figure 26:** A notional architecture bat chart of manned Mars exploration representing the primary viewpoints, OV and SV, of an architecture defined by DoDAF 2.0 [78, 115].

Morphological analysis is a leading technique for decomposing problems which have minimal design knowledge, such as those during the early phases of design [104]. Through morphological decomposition, an architecture space is defined which consists of the potential discrete options that exist in the physical architecture as a collection of parameters, along with options within each parameter. A matrix of alternatives (MOA), similar to the example shown in Figure 27, is typically used to represent the architecture space.

Recall from Section 2.1.6, an architecture is defined as the fundamental organization of a system of systems embodied by its systems and their relationships to each other and the environment. A system is defined as a set of regularly interacting subsystems, as stated in Section 2.1.1. Finally, a technology, as defined in Section



Interactive Reconfigurable Matrix of Alternatives (IRMA)

TOPSIS

| Platform | | | | |
|---|---|---|---|---|
| Platform | Presets | B-52 | B-1B | B-2 |
| | | F/A-22 | New Design | |
| | Cruise Speed | Subsonic | Supersonic (Yes) | Hypersonic |
| | Engine Type | Turbofan | Turbojet (No) | Ramjet |
| | | Pulse Detonation | Combined Cycle | Other |
| | Number of Engines | 1 | 2 | 4 |
| | Ferry Range | <1000 nm | 1000-3000nm (Yes) | 3000-5000 nm |
| | Refuelable | Yes | No | |
| | Piloting | Manned | Unmanned/Remote | Unmanned/Autonomous |
| | Stores | External | Internal Exposed | Internal Enclosed |
| | Wing Morphing | None | Variable Sweep | Variable Camber |
| | Body Style | Blended Wing | Flying Wing | Conventional |

| | | | | |
|---|---|---|---|---|
| Missile | Presets | Air Launched Tomahawk | JASSM | ASDL Parametric Model |
| | Primary Engine Type | Turbofan | Turbojet | Ramjet (Yes) |
| | | Rocket | Airbreathing Rocket | Pulse Detonation |
| | Inlet Position | Chin | Nose | Bottom |
| | | Twin Symmetric | None | |
| | Flight Speed | Subsonic | Supersonic | Hypersonic |
| | Range | < 300nm | 300-600nm | 600-1200 nm |
| | Wings | Subsonic Wings | Supersonic Wings | Hypersonic Wings |
| | Trajectory | Terrain Following | Low Altitude | High Altitude |
| | Controls | Tail | Canard | Thrust Vectoring |
| | Seeker/Guidance | Laser | Infrared | RADAR |

Possible Combinations: 2,519,424,000

Computational Analysis Time

| | |
|---|---|
| One Run per Second: | 79.89 Years |
| One Run per Minute: | 4,793.42 Years |
| One Run per Hour: | 287,605.48 Years |

Minimum TRL: 1

Air Force Asset Only: No

| | Continuous Input Variables: | | Discretizations per Variable: | |
|---|---|---|---|---|
| Platform | | 9 | | 6 |
| Missile | | 10 | | 7 |

Total Runs Required (Full Factorial): 737,261,740,903,680,000

Traditional Computational Analysis Time

| | |
|---|---|
| One Run per Second: | 2.34E+10 Years |
| One Run per Hour: | 8.42E+13 Years |

Total Runs Desired: 10,000,000

RSE Computational Analysis Time

| | |
|---|---|
| 1 Run per Second: | 115.74 Days |
| 10K Runs per Second: | 16.67 Minutes |

**Figure 27:** A notional Interactive Reconfigurable Matrix of Alternatives (IRMA) representing the architecture space [25].

2.1.5, is a device or subsystem developed to enable a specific capability. Through these definitions, technologies clearly operate on or as part of the physical architecture and appear as parameters with options during morphological decomposition. Unfortunately, this will only exacerbate the problem of combinatorial explosion, to be addressed later in this chapter. However, these definitions lead to the following conjecture:

**Conjecture 1.1**

Based on the definitions of an architecture and technology provided by this body of work, technologies affect an architecture by acting at the subsystem level of that architecture.

The design space consists of all design attributes and their feasible ranges. Typically, these become known once an analysis environment has been selected, as these design attributes are typically the inputs of the analysis environment. Groups of ranges for a given design attribute can be combined to form an overall range for that design attribute. Figure 28 is a notional representation of a design space.

The objective space is an n-dimensional space consisting of a collection of metrics which define the overall "goodness" of an architecture. An individual architecture is represented as a single point in the objective space. Typically, these metrics are values such as performance, cost, schedule, reliability, and other figures of merit. Figure 29 is a notional representation of a design space. Determining the specific objectives to include in the objective space will be discussed further in Section 3.3.1.3 and Section 3.3.3.

##### *3.3.1.2 Mapping System Spaces*

Consider the notional set of system spaces in Figure 30. Here, the architecture space parameters are mapped to at least one of the design space attributes. STSD defines these spaces as sets, and as such, the transformations are mappings from one set to another. The mappings, $f_{P_A}$ and $f_{P_B}$, translate parameter options in the architecture space to values or subsets of the design space attributes. There is no limit to the number of design space attributes a parameter is mapped to. These mappings may be considered a form of transformation, specifically, a step function. For example, consider $P_B$ in Figure 30 to be the type of engine(s), consisting of two options, $P_B^1$ as liquid oxygen engine(s) and $P_B^2$ as storable engine(s). $P_B$ is mapped to the design space attributes $D_B$ and $D_C$, representing specific impulse and boiloff rate, respectively. The mapping $f_{P_B}$ would then be a step function of the form:

$$f_{P_B} = \begin{cases} D_B^1 \cup D_C^1 & \text{if } P_B^1 \\ D_B^2 \cup D_C^2 & \text{if } P_B^2 \end{cases} \tag{1}$$

where $D_B^1$ may define a subset of $D_B$ in the range $(300 : 450)$ seconds, $D_B^2$ may

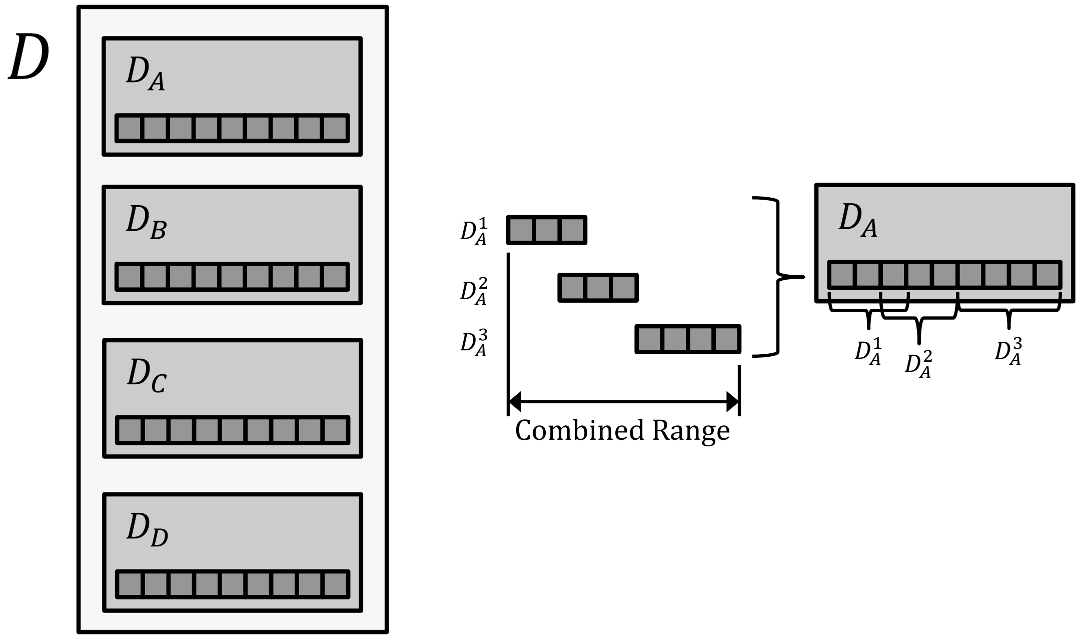


**Figure 28:** A design space consists of a collection of design attributes. Each design attribute may be a group of ranges to form an overall range for that attribute [104].

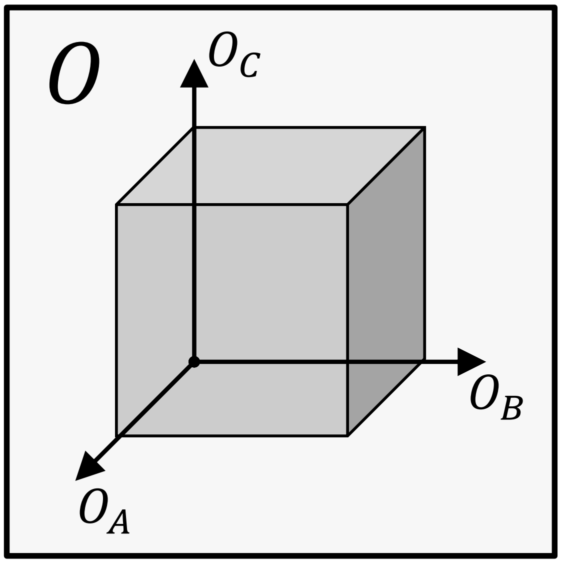


**Figure 29:** A notional three-dimensional objective space consisting of three independent objectives [104].

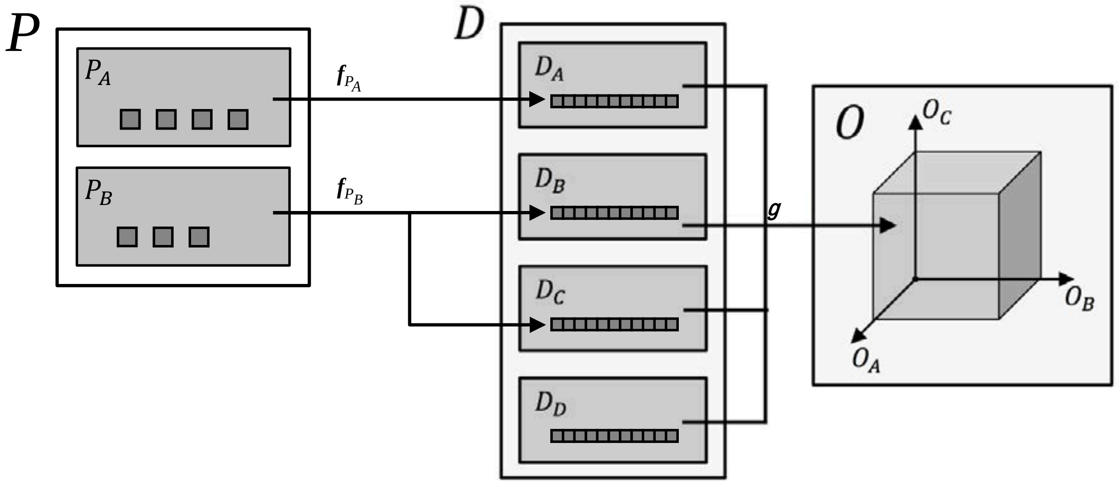


**Figure 30:** A notional mapping of the system spaces as described in STSD. The mappings between the architecture space and design space take the form of step functions where each architecture space option is mapped to at least one subset of a design space attribute. The mapping between the design space and the objective space is typically performed by a modeling and simulation environment which evaluates the set of design space attributes to provide the specific objective space metrics.

define a subset of $D_B$ in the range (275 : 350) seconds, $D_C^1$ may define a subset of $D_C$ in the range (5.0 : 10.0) kilogram of propellant loss per day, and $D_C^2$ may define a value of $D_C$ of 0.0 kilogram of propellant loss per day. This also provides an example of confounding of the design space. Here, specific impulses in the range (300 : 350) seconds cannot be strictly identified as being derived from the liquid oxygen or storable engine types. This effect can also be seen in Figure 28 where design attribute $D_A$ consist of overlapping subranges $D_A^1$ and $D_A^2$. Each of these subranges may be mapped to individual architecture parameter options. It would be impossible to know which architecture option a value in the overlapping design space attribute subset maps to.

In the notional example of engine type mapped to specific impulse, the selection of the range of specific impulse values mapped to an engine type is very subjective in nature, defined by a subject matter expert, systems engineer, or architect. The mapping between the design space and the objective space is typically much less subjective in nature, but also less transparent. Rather, the mapping between the design space and the objective space is typically fulfilled by a modeling and simulation environment. The relationship takes the form:

$$g = O(D) \tag{2}$$

The modeling and simulation environment is selected or developed for the purpose of translating design space attributes into objective space metrics. For example, if one of the objective space metrics is inert mass of the design, a modeling and simulation environment may utilize inputs such as specific impulse, delta-V, and mission duration as design attributes to calculate the inert mass. Obviously, complex systems of systems will contain many systems being sized simultaneously with multiple figures of merit in the objective space, requiring much more complex modeling and simulation environments. However, the idea that the modeling and simulation environment acts as the mapping between the design space and the objective space still holds.

The discussion of mapping system spaces up to this point has not taken into consideration the concept of technologies. This dissertation defines technologies at the subsystem level. Typically, these technologies will act on a design by modifying the inputs or outputs of an analysis environment. This technique of infusing technologies into a design is typically referred to as K-factor analysis. This is the technique utilized by many of the technology evaluation methods discussed in Section 2.4. However, to maintain a broad scope and flexibility during concept exploration, the way technologies are infused into the design should not be limited to just a single technique. Figure 31 illustrates a potential concept for introducing technologies into the system decomposition and mapping derived from STSD. Here the basic concept presented earlier is maintained, but with the addition of a new system space, the technology space, $T$. The technology space should contains all technologies identified during problem formulation. Each technology contains two options, active and inactive. The performance of a technology and its effect on a design are accounted for through the

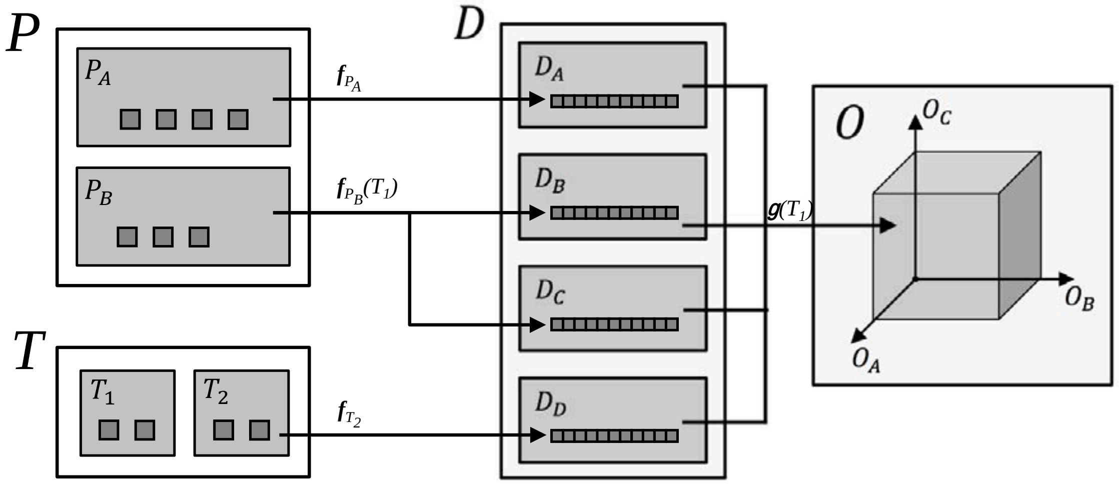


**Figure 31:** A modified notional mapping of the system spaces as described in STSD, with the addition of a technology space. Note that the mappings between the architecture space, design space, and objective space are now functions of the technology space.

mappings between the architecture space to the design space and the technology space to the design space. The notional system spaces in Figure 31 illustrate the different ways in which technologies may affect a design. For instance, the mapping $f_{P_B}$ is now a function of the technology $T_1$. However, technologies are not limited to interacting with a design through architecture space parameters mapped to design space attributes. Technologies may interact directly with design space attributes, as is indicated by the mapping $f_{T_2}$. Finally, the effect of technologies may also interact with the mappings between the design space and the objective space, indicated in Figure 31 as $g(T)$.

To better illustrate these concepts, the example stated earlier shall be revisited. Recall the mapping of architecture space parameter $P_B$ representing type of engine(s) to the design space attributes $D_B$ and $D_C$, representing specific impulse and boiloff rate. Now define the technology space parameter, $T_1$, as boiloff mitigation. The mapping $f_{P_B}$ now takes the following form:

$$f_{P_B}(T_1) = \begin{cases} D_B^1 \cup D_C^1 & \text{if } P_B^1 \cup T_1^0 \\ D_B^2 \cup D_C^2 & \text{if } P_B^2 \cup T_1^0 \\ D_B^3 \cup D_C^3 & \text{if } P_B^1 \cup T_1^1 \\ D_B^4 \cup D_C^4 & \text{if } P_B^2 \cup T_1^1 \end{cases} \tag{3}$$

The subranges $D_B^1$, $D_B^2$, $D_C^1$, and $D_C^2$ all maintain the same definition as described before, as these would be the design attributes when the technology $T_1$ is inactive, indicated by $T_1^0$. The subranges $D_B^3$, $D_B^4$, $D_C^3$, and $D_C^4$ would then represent the design attributes with technology $T_1$ active, indicated by $T_1^1$. For instance, if the technology has no effect on the specific impulse, $D_B^3$ and $D_B^4$ would maintain the same ranges defined by the inactive technology scenario. However, $D_C^3$ and $D_C^4$ may define a value of 0.0 kilogram of propellant loss per day due to the active technology. Furthermore, Figure 31 shows a direct mapping between the technology space and the design space, seen by the mapping $f_{T_1}$. This could be utilized in a scenario where a technology affects the design through an attribute which has no mapping from the architecture space. For instance, consider technology $T_2$ to be low leak valves for engines and design space attribute $D_D$ to be propellant leakage of the engine. Here, propellant leakage is unaffected by any architecture space parameters, but is directly affected by the technology $T_2$. The mapping $f_{T_2}$ would have the form:

$$f_{T_2} = \begin{cases} D_D^1 & \text{if } T_2^0 \\ D_D^2 & \text{if } T_2^1 \end{cases} \tag{4}$$

Where $D_D^1$ may be the a leak rate of 50 kilograms of propellant per engine start and stop cycle and $D_D^2$ may be a zero leak rate of propellant per engine start and stop cycle. Again, the values associated with these subsets are subjective, determined by the technologies, systems engineer, architect, etc. This notional example considered fixed ranges and values. However, these subsets may be estimated utilizing K-factor analysis on nominal design attribute ranges, through functions of the specific architecture and technology space options, or other techniques for infusing the effect of technologies on design attributes.

The notional set of system spaces also shows the mapping between the design space and the objective space to be a function of $T_1$. This would then take the form:

$$g = O\left(D, T_1\right) \tag{5}$$

As an example, assume the objective $O_A$ represents the inert mass of the design and the mapping $g\left(T\right)$ represents the modeling and simulation environment as described earlier. If the activation of technology $T_1$ results in a mass growth of the burnout mass of the design, then an element of the mapping $g\left(T\right)$ may take the form:

$$O_A = \begin{cases} m_{bo}\left(D\right) + m_{prop}\left(D\right) & \text{if } T_1^0 \\ k_1 * m_{bo}\left(D\right) + m_{prop}\left(D\right) & \text{if } T_1^1 \end{cases} \tag{6}$$

Where $k_1$ represents the effect due to $T_1$. This value would again defined by the technologist, systems engineer, or architect. The element $O_A$ of the transformation shown utilizes a form of K-factor analysis to evaluate technologies. However, this is not a strict requirement, but rather an illustration used for the purpose of explaining the basic concept of mapping the design space to the objective space. In reality, these mappings may take a variety of forms.

Utilizing STSD, a new system space, the technology space, may be added which contains the technologies and their settings. These are derived during problem formulation and decomposition of the problem at the same time, and in a similar manner as, the architecture space. This modification to STSD allows the decompositions of complex system of systems spaces which define and integrate technologies and architectures at the subsystem level. This leads to the following conjecture:

**Conjecture 1.2**

Based on the assumed decomposition derived from STSD, a technology space should be included as a new system space. The transformations between the architecture space, design space, and objective space shall be functions of the new technology space.

#### *3.3.1.3 Establishing Value*

Once a problem has been defined and decomposed, the final step in characterizing the trade space requires establishing value for the trade space. This value will be determined by the metrics chosen as measures of "goodness" of the designs. In Section 2.4, several technology evaluation methods were presented which assign value to a design in various ways. Table 5 summarizes these methods and the approaches taken for evaluating technologies. Of the five technology evaluation methods presented, TIES, ATIES, TRIPS, and TAPP represent technologies as a subsystem of the design, which aligns with how technologies are considered in this body of work. Collectively, these methods evaluate technologies on the basis of performance, cost, schedule, and organizational alignment. However, no single method evaluates technologies across all of these metrics.

Additionally, it is known that technologies may have a profound impact on many of these metrics. Such observations have led to the development of metrics to assess an overall readiness of a design as a function of the maturity of incorporated technologies and their respective links between other systems and technologies [99]. This leads to the following research question:

**Research Question 2**

What technology impact metrics should be considered in determining the overall "goodness" of the results such that they may be ranked?

Because technologies impact an architecture at the subsystem level and interact with the analysis environment through the system space transformation functions, technologies can only interact with metrics which already exist within the architecture analysis environment. The metrics of interest for architecture analysis are determined based on the customer's desires and trade studies of interest. Furthermore, it has been argued that metrics developed for assessing the readiness of designs, such as the System Readiness Level (SRL) developed by Sauser et al., may be misleading in meaning. In his analysis of Sauser's SRL, Edouard Kujawski concludes the following [62]:

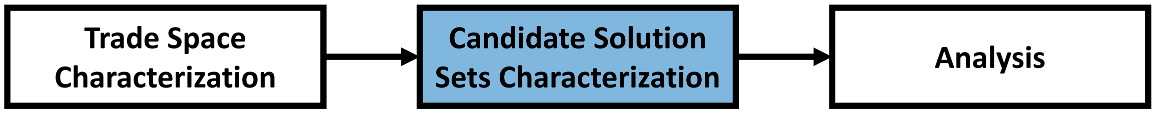
1. It utilizes invalid arithmetic operations on ordinal data
2. Inputs do not provide information on risk and effort required for achieving higher readiness levels
3. Filters out microscopic information needed in managing specific risk areas

These observations supports the requirement to be multi-objective in nature, not relying on a single utility metric or readiness value. Typically high-level design studies are not performed by a specific subject matter expert or technologist, but rather a systems engineer or architect. These individuals are generally less biased towards any one architecture or technology. However, they are still people and will impose a level of bias on the study. Transparency throughout the design decomposition and evaluation processes helps to alleviate this bias. However, the validity of potential metrics for evaluating the overall "goodness" of designs is not within the scope of this research. The task of identifying techniques to reduce bias in the selection of technologies is left for future research. This leads to the following conjecture:

**Conjecture 2**

The technology impact metrics considered should align with the metrics of interest for measuring the "goodness" of an architecture, without introducing unnecessary bias.

### 3.3.2 Phase II: Candidate Solution Sets Characterization


Trade Space Characterization
Candidate Solution Sets Characterization
Analysis

The second phase of concept exploration requires characterizing candidate solution sets, otherwise known as alternatives. When evaluating technologies on the architecture scale, new concerns arise unique to problems at this level. Typically, technology evaluation is performed only after a baseline design is selected, optimized, and determined infeasible or inviable without incorporating new technologies. This paradigm is prevalent in modern methods for architecture analysis and technology evaluation. Many of the methods presented in Chapter 2 require selecting and defining a baseline design. However, during the early phases of design, and indeed, following the decomposition of the problem as described in Section 3.3.1, many different architectures exist which must be evaluated. This leads to the following research question:

**Research Question 3**

Is the paradigm of down-selecting to a baseline design on which to perform technology analysis sufficient for the exploration of complex architectures?

A simple, notional problem shall help formulate a hypothesis. Consider the problem of selecting the best in-space propulsion stage. The objective is to minimize inert mass of the design, measured in kilograms. Assume there are two competing architectures, a storable propellant propulsion module and a methane propellant propulsion module. There is also a propellant boiloff mitigation technology which may be infused into the design. This results in a total of four alternatives to be evaluated:

1. Storable Propulsion Module
2. Methane Propulsion Module
3. Storable Propulsion Module with Boiloff Mitigation Technology
4. Methane Propulsion Module with Boiloff Mitigation Technology

The architecture space contains two parameters, mission and propulsion system. The mission is assumed fixed with only one option. Propulsion system contains two options, storable or methane. All alternatives will assume the same burnout mass of 2000 kilograms and an identical mission definition with a transit of 330 days followed by a single impulsive $\Delta$V burn of 4,000 meters per second. The technology space consists of a single technology, a boiloff mitigation technology which eliminates propellant boiloff with a 5% growth in burnout mass. The design space consists of four parameters, impulsive delta-V, mission duration, specific impulse, and boiloff rate. The objective space consist of a single metric, inert mass. Figure 32 provides a graphical representation of the system spaces and mappings for this notional example. The mappings $f_{M1}$ and $f_{PropSys}$ take the following forms:

$$f_{M1} = D_{\Delta V} \cup D_{\Delta T} \tag{7}$$

$$f_{PropSys}\left(T\right) = \begin{cases} D^1_{I_{sp}} \cup D^1_{BOR} & \text{if } P^{Storable}_{PropSys} \\ D^2_{I_{sp}} \cup D^2_{BOR} & \text{if } P^{Methane}_{PropSys} \\ D^1_{I_{sp}} \cup D^3_{BOR} & \text{if } P^{Storable}_{PropSys} \cup \text{Boiloff Mitigation} \\ D^2_{I_{sp}} \cup D^4_{BOR} & \text{if } P^{Methane}_{PropSys} \cup \text{Boiloff Mitigation} \end{cases} \tag{8}$$

$D_{\Delta V}$ and $D_{\Delta T}$ are defined as 4,000 m/s and 330 days, respectively. $D^1_{I_{sp}}$ is defined as 300 seconds and $D^1_{I_{sp}}$ is defined 350 seconds. Finally, $D^1_{BOR}$ is defined as 5 kilograms of propellant loss per day, while $D^2_{BOR}$, $D^3_{BOR}$, and $D^4_{BOR}$ are defined as zero kilograms of propellant loss per day. The mapping from the design space to the objective space, $g\left(T\right)$ is represented by equations 9 - 12.

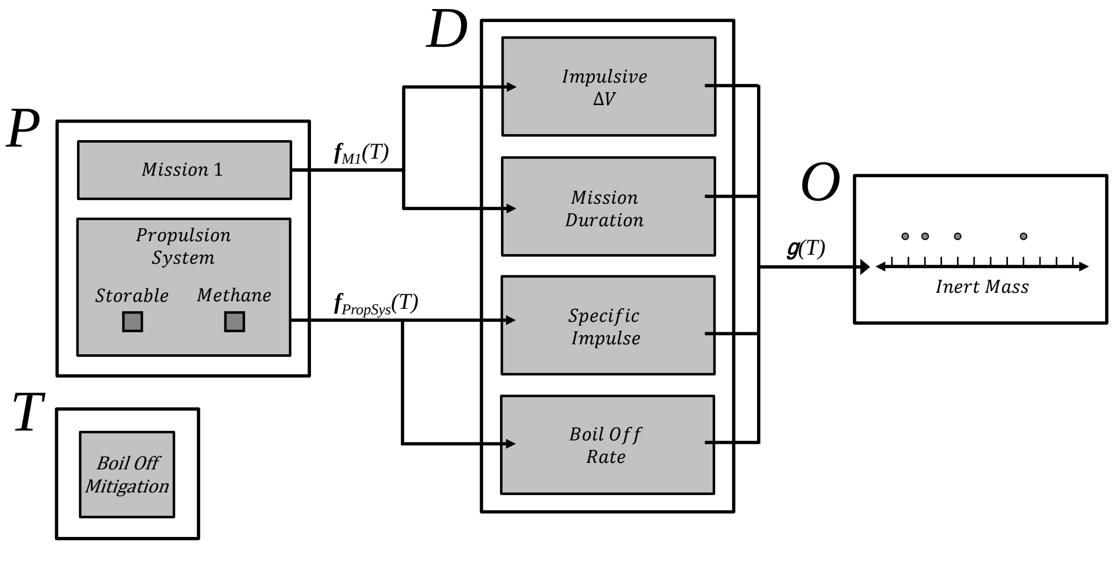


**Figure 32:** Notional Example of a Simple Space Transportation Architecture Problem

$$m_{burned} = m_{burnout}(e^{\Delta V / g_0 I_{sp}} - 1) \tag{9}$$

$$m_{boiled} = R_{boil} \Delta T \tag{10}$$

$$m_p = m_{burned} + m_{boiled} \tag{11}$$

$$m_{inert} = \begin{cases} m_p + 1.05 * m_{burnout} & \text{if Boiloff Mitigation} \\ m_p + m_{burnout} & \text{otherwise} \end{cases} \tag{12}$$

**Table 6:** Notional Example Objective Results

| **Architecture** | **Inert Mass(kg)** |
|---|---|
| Storable | 7789.4 |
| Methane | 8064.3 |
| Storable w/ Tech | 7889.4 |
| Methane w/ Tech | 6514.3 |

Analysis of the architectures results in performance values as seen in Table 6. Following the paradigm of optimizing a design before analyzing technologies, the storable architecture would be selected for technology evaluation with a performance of 1.712 over the methane architecture's performance of 1.548. However, the methane architecture with technology enhancements performs better than any other architecture, at 2.079. Following the traditional paradigm of optimizing a design before performing technology evaluation results in this architecture being overlooked. This occurred because under the traditional technology evaluation paradigm, it is assumed that the impact of a technology on the design is consistent across the alternatives being optimized. However, in the design of architectures, the underlying systems are not consistent. Since technologies act upon these underlying systems, the overall effect of technologies on architectures is not consistent across designs. This notional example leads to the following hypothesis:

**Hypothesis 3**

The paradigm of down-selecting to a baseline design and then performing technology analysis will not be sufficient in performing architecture design. This paradigm assumes that the systems a technology acts upon remain constant throughout the down-selection process. Because these systems vary between architectures, the effects of technologies will be inconsistent among these architectures.

The notional example just discussed showed that the design of complex architectures and technologies will require many additional design alternatives to fully characterize the problem because technologies must be evaluated for each alternative before down-selection. The large combinatorial space will likely lead to prohibitively long analysis times. This leads to the following research question:

**Research Question 4**

How is combinatorial explosion of the number of alternatives affected by different types of system spaces?

To overcome challenges due to large numbers of alternatives due to combinatorial explosion, Design of Experiments (DOE) may be used. DOE is a technique by which a set of experiments is selected to maximize information while minimizing experimental effort [65]. This technique lends itself to various uses:

1. **Comparative:** Assessing the impacts on the process as a whole as a result of changes in a single input factor

2. **Characterization:** Understanding the importance of various input factors on the process as a whole

3. **Modeling:** Obtaining input/output sets in an attempt at estimating a process

through some mathematical function

4. **Optimization:** Determining optimal settings of the input factors to obtain an optimal process response

There are many different ways of generating these “sets” of experiments. Before the widespread use of computers, Taguchi utilized orthogonal arrays to determine sets of experiments. However, today, software packages such as JMP® Statistical Discovery from SAS can create custom designs quickly and easily. Historically, DOE was developed as a method for creating sets of inputs for physical experiments in an effort to minimize randomness in the sets such that responses to changes in those inputs could be accurately estimated. Another term used specifically for creating sets of experiments when working with computational experiments is Design of Computer Simulations (DoCS). The goal of DoCS is similar to DOE, obtaining maximum information through minimal effort. However, the sets of experiments are tailored to be more suitable for deterministic computational analysis tools. The most notable difference between physical experimentation and computational experimentation is the number of factors involved. Typically computational experiments contain many more factors for testing [67]. In practice, DoCS and DOE are generally understood to have similar meaning and may be used interchangeably. System spaces can be broken into one of three types:

1. **Continuous/Discrete Ordinal:** All input variables can be ordered in the ranges being considered

2. **Discrete Categorical:** All input variables cannot be ordered in the ranges being considered

3. **Mixed:** Both ordinal and categorical input variables exist in the ranges being considered

An architecture alternative space will almost always be a mixed space, containing both ordinal and categorical design variables. A simple example proves this to be self-evident. Consider the design of a single stage rocket with three design variables: engine propellant, engine specific impulse and number of engines. Engine propellant options are discrete categorical in nature because there is no inherent ordering to propellant types such as LOX/Methane versus Storable. However, engine specific impulse is a continuous ordinal variable because the values that represent the variable have inherent order. For example, a specific impulse of 350 seconds is greater than 300 seconds. Finally, the number of engines is a discrete variable but is ordinal in nature. Three engines is greater than two engines, but a design may not contain 2.5 engines. Mixed design spaces are typically understood to require a full factorial DOE on the categorical variables, while other DOEs can be used for the ordinal variables [67]. However, because of the large, complex nature of architecture design problems, there exist prohibitively large numbers of alternatives to consider. Performing a full factorial DOE on even a portion of the design space can be a challenge.

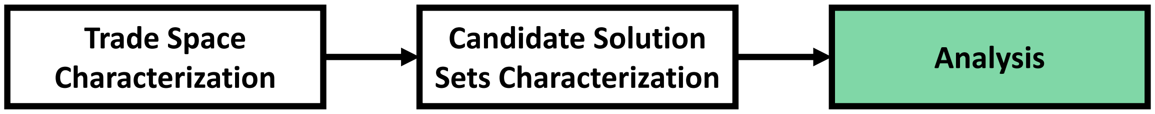
Suppose a vehicle space is defined by a vehicle having up to four stages. Each of these stages is described by five discrete parameters with two levels each. This results in millions of unique vehicle definitions. Likewise, if we consider a mission space defined by ten unique missions, each described by 5 individual events, with each event having four discrete parameters containing two levels, this produces on the order of tens of millions of unique mission definitions. If an architecture consists of a unique mission-vehicle combination, then there are on the order of $10^{13}$ possible architecture alternatives in this notional architecture space. Remember that the vehicle and mission definitions only contain the discrete variables and do not account for any continuous variables which may exist in a mixed space. This means in our objective space, each of these architectures would appear as a single point, but in reality, is a cloud of points dependent on the ranges of the continuous variables of the design. However, because the continuous variables are not considered in this example, they are assumed to take default values, collapsing the cloud of points to a single point in the objective space. This act of selecting default values for the continuous space may be a viable solution to reducing the overall number of potential designs which must be analyzed. However, study of this option is outside the scope of this body of work.

Also note, our architectures do not account for technologies in any way, which will further increase the complexity and number of design alternatives. If it is assumed that an architecture analysis environment exists that can perform its task in one second per architecture per central processing unit (CPU) core, then it would take 44,984 years to complete analysis of all alternatives in this notional design space on a standard 8-core personal computer. Even utilizing all 246,048 CPU cores of the Pleiades supercomputer would take 1.46 years to analyze all the alternatives. This phenomenon is what is referred to as combinatorial explosion. The typically understood assumption of requiring full factorial DOEs for the discrete categorical variables of a mixed design space, on top of other DOEs for the ordinal variables, is impractical for a realistic design study. However, minimizing combinatorial explosion is outside the scope of this research. To minimize the effect due to combinatorial explosion, the following conjecture is made:

**Conjecture 4**

Subsets of the discrete architecture and technology space parameters will be selected, along with assumed values for continuous parameters, dependent on the discrete options, to minimize combinatorial explosion such that effects due to integrating technologies and architectures at the subsystem level in a single method can be studied.

### 3.3.3 Phase III: Analysis


Trade Space Characterization
Candidate Solution Sets Characterization
Analysis

The final step in the general concept exploration framework consists of analyzing the alternatives. The end results of analysis are numerical values of the figures of merit selected for ranking designs. These figures of merit will help decision makers come to final conclusions regarding the overall design. As was shown in the previous section, large numbers of alternatives exist in such a complex architecture space. Care must be taken to ensure that useful results are observable among so much data. Including technologies into the architecture design problem only complicates the presentation of results by creating ever larger numbers of alternatives as well as figures of merit to consider. This leads to the following research question:

**Research Question 5**

How shall results be presented to allow decision makers clear, concise choices in selecting architectures and technologies?

#### *3.3.3.1 Individual Architecture Scheme*

In an individual architecture scheme, each alternative analyzed is presented in the final results. With a potential for billions of alternatives or more, it is not difficult to imagine subtle details in trends of the results being washed out and difficult to observe with so much data to present. This leads to the following research sub-question:

**Research Question 5.1**

Would utilizing an individual architecture presentation scheme prevent high-level effects of architecture design decisions from being observed?

The highly discrete nature of architecture design means that there will more than likely be grouping of the results corresponding to discrete decisions in the matrix of alternatives. If a given option in the matrix of alternatives drives extraordinary results for a subset of the alternatives, that subset of alternatives would be ranked higher than any of the other subsets. However, during early phases of design, decision makers are typically more interested in how architecture design decisions, such as applying a certain technology or basic propulsion types, may affect the performance of an architecture compared to the others. However, it could be challenging to see this type of information with so many results of a subset of alternatives flooding the top rankings of the results. In essence, by narrowing the amount of data being observed by only relying on a top N individual architectures scheme, a poor cross section of the whole objective space will be observed.

As an example, suppose a decision maker is interested in understanding how the selection between four different propulsion systems may affect the resulting architectures. Assume there are 1000 architectures in the space, 250 for of each propulsion type, and that the top 10 performing individual architectures will be presented. It is probable that all 10 top architectures may be of a single propulsion type. However, if the decision maker wishes to perform comparative analysis of the different propulsion types in the objective space, this information cannot be observed due to only a single propulsion type from the architecture space flooding the chosen objective space. The top 10 architectures resulted in an objective space with a poor cross section of the original architecture space. This logic leads to the following hypothesis:

**Hypothesis 5.1**

The presentation of individual architectures will obscure high-level effects due to flooding of the top results with similar individual designs.

#### 3.3.3.2 *Portfolio Scheme*

In the previous section, it was observed that selecting optimal designs from an objective space containing individual architectures may result in a poor cross-section of the architecture space actually being studied and observed. However, because the architecture space is so discrete in nature, portfolios of designs may be formed to observe cross sections that are more representative of the architecture space. In a portfolio scheme, results of the alternatives may be grouped in some manner in an attempt to simplify the presentation of results in situations where there is an unmanageable amount of individual data points to present. In his work on developing a rapid architecture analysis model, Iacobucci states that common tools used for data exploration are not designed to handle extremely large data sets [50]. In his work, portfolios were created based off of unique sets of systems. Architectures containing the same constituent systems were grouped together into portfolios. Similarly, methods such as TIES use sets of active technologies to act as grouping criteria for the objective space. However, details about how grouping criteria are selected in these methods are undefined. There has been no study of how these grouping criteria may affect the resulting portfolios. This leads to the following research question:

**Research Question 5.2**

How do the grouping criteria used for forming portfolios of architectures affect the variance of the resulting portfolios?

The concept behind this research question is illustrated by Figure 33. Here, grouping criteria can be selected in many ways to either form a few large portfolios or many small portfolios spanning the objective space.

If portfolios are defined in such a way that they contain large numbers of architecture alternatives, it is expected that there will be many different architecture concepts

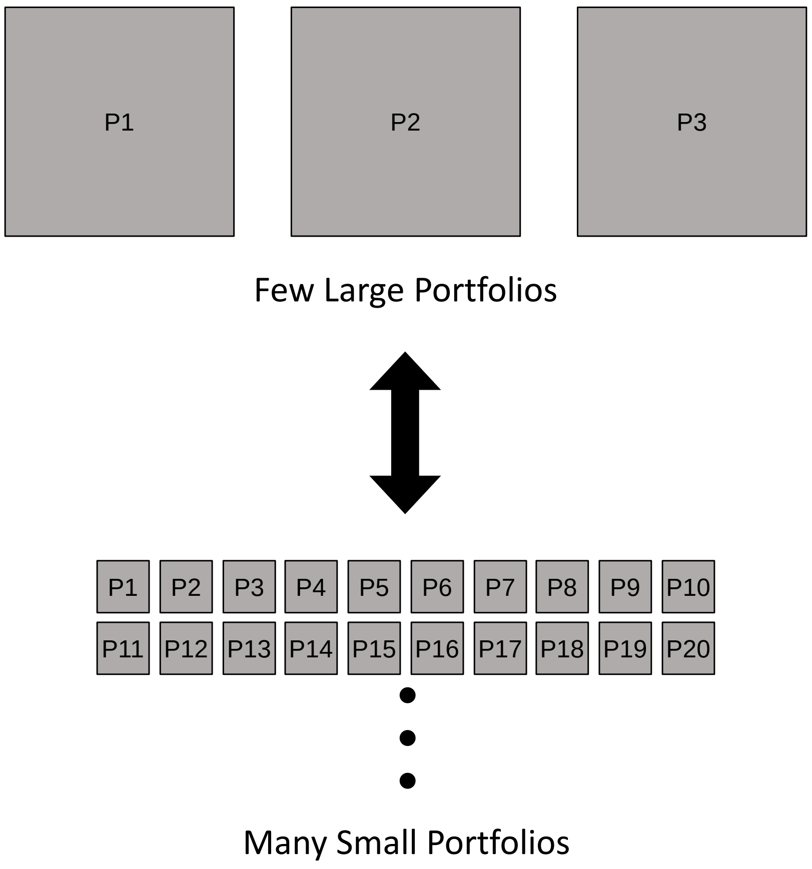


**Figure 33:** Relationship Between Portfolio Size and Number of Portfolios

grouped together, creating a large variation in architectures within a given portfolio. If the individual architecture objectives are rolled up into portfolio-level objectives, it is expected that the variations in these rolled up metrics will be greater compared to if the objective space were broken into many smaller portfolios. However, when the scope of an individual portfolio is narrowed down, the number of possible portfolios increases. With more focused architecture grouping criteria, variation between the grouped architectures in a portfolio will be reduced compared to the large portfolios. These ideas are summarized in the following hypothesis:

**Hypothesis 5.2**

Variance of the objective metrics within and between portfolios will correlate positively with the size of the portfolios, measured by the number of grouped architectures.

However, the act of grouping criteria together to form portfolios of architectures spanning the objective space may leads to potential problems with observing results and selecting optimal designs. Because architectures are grouped together with the performance of portfolios being compared as opposed to individual designs, there may exist a scenario where a high-performing optimal design is obscured in a lower-performing portfolio, formalized in the following research question:

**Research Question 5.3**

Would utilizing a portfolio scheme for grouping architectures obscure high-performing outlier architectures?

As a notional example, consider the simple single objective space containing 20 total designs, represented by Table 7. Two portfolios of designs are created with the simple grouping criteria of the first 10 designs and the remaining 10 designs. Here we see the optimal design, denoted by the maximum objective value, resides in portfolio two. However, by creating a portfolio-level metric as the mean of the 10 contained design objective values, portfolio two has an overall objective value of 0.46. The same overall objective for portfolio one is 0.67. If a decision is made to only consider designs contained within the optimal portfolio, the true optimal design is obscured within the suboptimal portfolio. In order to more easily develop an experiment around this research question, a null hypothesis will be set up. This leads to the following:

**Table 7:** Notional Design Obscuring In Objective Space Portfolios

| | Objective | |
|---|---|---|
| Designs | 0.766211 | P1=0.67 |
| | 0.804294 | |
| | 0.759173 | |
| | 0.730905 | |
| | 0.574270 | |
| | 0.620439 | |
| | 0.618472 | |
| | 0.654051 | |
| | 0.245305 | |
| | 0.914338 | |
| | 0.372797 | P2=0.46 |
| | 0.654160 | |
| | 0.147452 | |
| | 0.087135 | |
| | 0.480864 | |
| | 0.328384 | |
| | 0.934477 | |
| | 0.916002 | |
| | 0.155315 | |
| | 0.498400 | |

**Hypothesis 5.3**

High-performing outlier architectures will not be obscured using a portfolio evaluation scheme because they will be contained in a portfolio with other similar architectures which will exhibit similar behavior, raising the performance of the entire portfolio.

#### *3.3.3.3 Effects on Establishing Value*

Due to the sheer number of potential alternatives which may exist in the concept exploration of complex architectures and technologies, presentation of the results will provide new challenges. The way in which the results are presented could dramatically alter the metrics of interest. This is formalized in the following research question:

**Research Question 6**

Does the presentation scheme of the results affect the metrics that should be utilized to establish the value of a portfolio?

At the core of all of the information available to a decision maker during architecture evaluation is the analysis environment. The environment dictates what information is available for use as figures of merit in ranking architectures or portfolios of architectures, as well as producing DoDAF-based viewpoints of the architecture. Because of this, it is not possible to obtain different sets of metrics Dependant on the presentation scheme. Rather, the set of metrics is fixed based on the analysis environment. Also, since a portfolio would be a grouping of similar architectures, it follows that the portfolio results will have the same figures of merit as those of the individual architectures. An aggregate of the metric can be calculated for an entire group of alternatives. However, by grouping architectures together, high-level architecture decisions can be observed through the variation in the individual architectures in a portfolio. Whereas presenting architectures independently can make it difficult to perform the required analysis to observe these effects, portfolios provide a logical, predefined subgroup to make studying high-level architecture choices concise. Furthermore, through the use of a multi-level Unified Tradeoff Environment (UTE) similar to the notional example shown in Figure 34, simultaneous trades between the architecture, design, objective, and technology spaces can be performed [10]. Weightings among these spaces and their attributes can be selected to study the effects on a portfolio scheme. This is summarized in the following hypothesis:

**Conjecture 6**

A portfolio scheme should include new figures of merit relating to the portfolio metric variances, portfolio composition, and portfolio objective weightings.

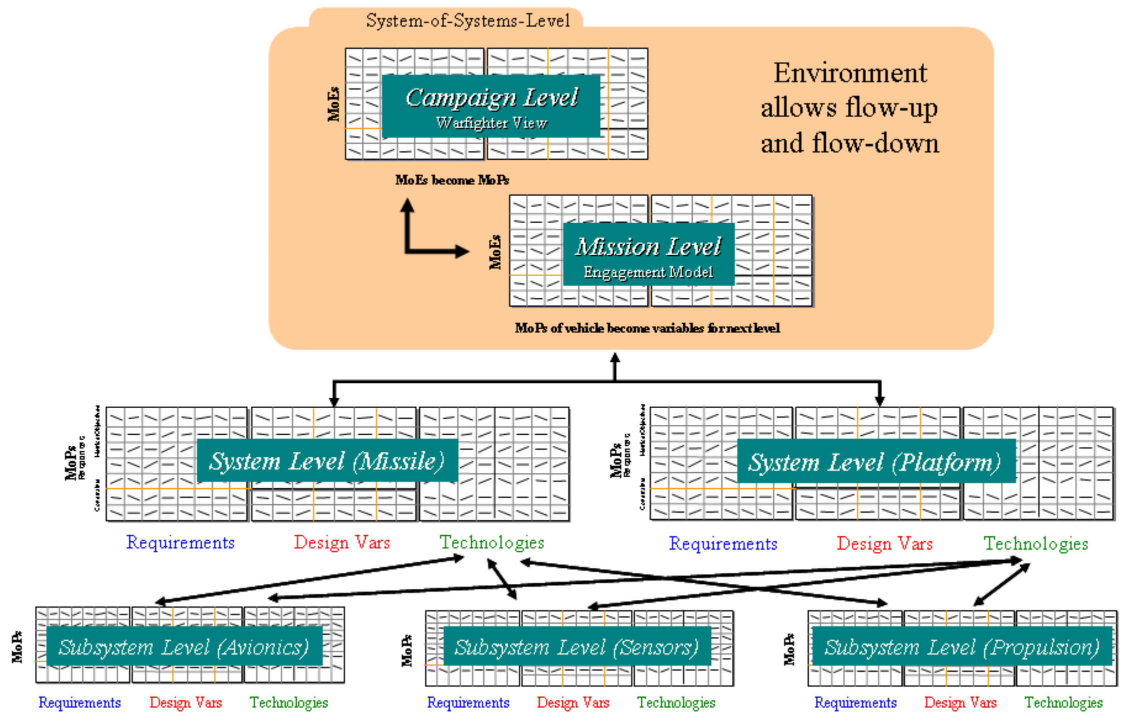


**Figure 34:** A notional multi-level Unified Tradeoff Environment (UTE) of forward ground support through strategic airlift architectures [10]

## *3.4 Summary of Research Questions*

This chapter has proceeded through a general process for concept exploration, and in doing so produced a set of research questions and hypotheses relating to integrating technology evaluation and architecture analysis into a single framework. The following is a summary of those questions and hypotheses:

**Statement of Purpose**

To provide a capability to analyze complex systems of systems to an extent which will provide decision makers in the early phases of design sufficient information to reduce the risks associated with cost and schedule overruns due to lack of design knowledge.

**Research Objective**

To integrate architecture analysis and technology evaluation at the subsystem level to provide a quantitative framework in an effort to increase design knowledge early in the design process.

**Research Question 1**

What is the relationship between architectures and technologies?

**Conjecture 1.1**

Based on the definitions of an architecture and technology provided by this body of work, technologies affect an architecture by acting at the subsystem level of that architecture.

**Conjecture 1.2**

Based on the assumed decomposition derived from STSD, a technology space should be included as a new system space. The transformations between the architecture space, design space, and objective space shall be functions of the new technology space.

**Research Question 2**

What technology impact metrics should be considered in determining the overall “goodness” of the results such that they may be ranked?

**Conjecture 2**

The technology impact metrics considered should align with the metrics of interest for measuring the “goodness” of an architecture, without introducing unnecessary bias.

**Research Question 3**

Is the paradigm of down-selecting to a baseline design on which to perform technology analysis sufficient for the exploration of complex architectures?

**Hypothesis 3**

The paradigm of down-selecting to a baseline design and then performing technology analysis will not be sufficient in performing architecture design. This paradigm assumes that the systems a technology acts upon remain constant throughout the down-selection process. Because these systems vary between architectures, the effects of technologies will be inconsistent among these architectures.

**Research Question 4**

How is combinatorial explosion of the number of alternatives affected by different types of system spaces?

**Conjecture 4**

Subsets of the discrete architecture and technology space parameters will be selected, along with assumed values for continuous parameters, dependent on the discrete options, to minimize combinatorial explosion such that effects due to integrating technologies and architectures at the subsystem level in a single method can be studied.

**Research Question 5**

How shall results be presented to allow decision makers clear, concise choices in selecting architectures and technologies?

**Research Question 5.1**

Would utilizing an individual architecture presentation scheme prevent high-level effects of architecture design decisions from being observed?

**Hypothesis 5.1**

The presentation of individual architectures will obscure high-level effects due to flooding of the top results with similar individual designs.

**Research Question 5.2**

How do the grouping criteria used for forming portfolios of architectures affect the variance of the resulting portfolios?

**Hypothesis 5.2**

Variance of the objective metrics within and between portfolios will correlate positively with the size of the portfolios, measured by the number of grouped architectures.

**Research Question 5.3**

Would utilizing a portfolio scheme for grouping architectures obscure high-performing outlier architectures?

**Hypothesis 5.3**

High-performing outlier architectures will not be obscured using a portfolio evaluation scheme because they will be contained in a portfolio with other similar architectures which will exhibit similar behavior, raising the performance of the entire portfolio.

**Research Question 6**

Does the presentation scheme of the results affect the metrics that should be utilized to establish the value of a portfolio?

**Conjecture 6**

A portfolio scheme should include new figures of merit relating to the portfolio metric variances, portfolio composition, and portfolio objective weightings.

# CHAPTER IV

# SPACE TRANSPORTATION ARCHITECTURE MODELING

Performing architecture and technology trades on space systems is a difficult problem because, by definition, they are system of systems problems as outlined in Chapter 2. The large system of systems gives rise to extremely large trade spaces suffering from analysis issues such as combinatorial explosion, as described in Chapter 3. In order to perform analysis on the large combinatorial trade space, there needs to be a way of quickly evaluating different architecture concepts. In the context of spaceflight, there are many new concepts that need to be modeled under various scenarios, usually utilizing physics-based analysis due to a lack of historical data to populate the vast design space. This chapter will investigate the basic analysis concepts, ontology, and existing tools in the domain of space transportation architecture analysis. A shortcoming in current tools will lead to a discussion of a new subsystem-level, multidisciplinary design, analysis, and optimization (MDAO) framework for space transportation architecting, as well as spacecraft subsystem models developed for the purpose of this body of work.

## *4.1 Basic Modeling and Simulation Concepts*

Before describing space transportation analysis and design, it would be advantageous to discuss a few basic concepts in modeling and simulation and how they apply to space transportation architecture analysis. The two concepts of interest in this discussion are surrogate modeling and optimization, as their implementation can have dramatic impacts on the performance of analysis tools. The goal of this section is to make the reader aware that these concepts were considered during the formulation and creation of the framework and models discussed later in Section 4.4 and Section 4.5. However, no formal research questions or hypotheses were formed with regard to basic modeling and simulation concepts, and is considered outside the scope of this body of work.

### 4.1.1 Surrogate Modeling

Surrogate modeling is a technique by which complex physics-based models are approximated by means of some independent mathematical construct such as response surface equations, neural networks, or even simple algebraic equations, to name a few. Literature provides a multitude of different surrogate modeling techniques. A discussion of the different modeling techniques and their applicability to systems of systems problems is outside of the scope of this research. The key principles behind these surrogate models are two-fold. The first is to speed up the process of evaluation. The second is the ability to obscure proprietary source codes as well as providing the ability to create frameworks that are tool independent [66]. Typically, complex multi-physics-based analysis tools are considered too slow to be utilized in an automated design framework, and many organizations are unwilling to make proprietary models open and available. Surrogate modeling then becomes a key enabler to creating fast and highly accurate models to be utilized in frameworks to aid in the early stages of design. It allows large numbers of designs to be evaluated and analyzed with relative ease and speed, while focusing only on relevant data. Typically, the process of creating surrogate models relies on utilizing a DOE to intelligently gather the large amounts of data from these complex, slow, proprietary analysis tools necessary to generate the surrogate models. Due to the discrete nature of architectures defined in this dissertation, creating surrogate models of the upper-level system of systems cannot be done. For architectures defined in this way, surrogate models are typically created around physics-based subsystem or component models. It is these surrogate models which are then brought together in an analysis framework to perform architecture design and analysis.

### 4.1.2 Optimization

Dieter defines optimization as "the process of maximizing a desired quantity or minimizing an undesired one." [22] In the context of numerical analysis of designs, optimization is typically a logical approach to design automation [117]. Typically, these logical approaches are in the form of algorithms which explore a design space methodically to reach a desired result. Again, literature provides a variety of algorithms for this purpose. However, an exploration of the many different optimization techniques is outside of the scope of this dissertation and is left for future work. Typically, these techniques are broken into two main types, local optimizers and global optimizers. Figure 35 provides a notional example of these differences. In this example, the optimum is a minimum. Obviously, one can observe that different optimizers can provide drastically different results in solutions, both at a subsystem and system of systems level.

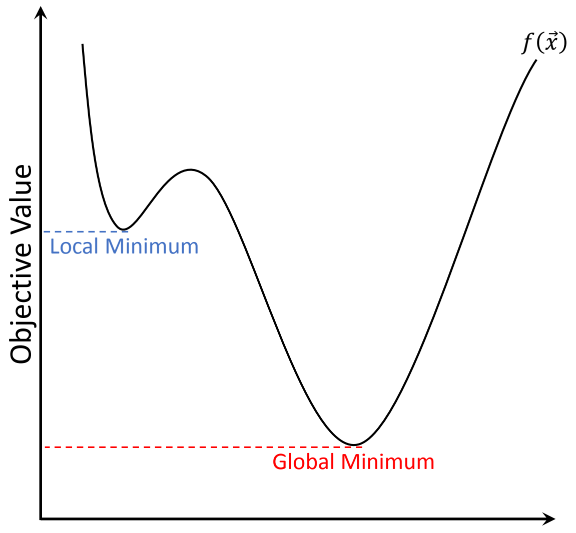


**Figure 35:** Notional Example of Local vs Global Minimum

The analysis of complex architectures requires multiple levels of optimization to achieve a final optimized result. This is due to the multi-level, multi-model structure of architecture design problems. In fact, there is an entire field dedicated to performing multi-level optimization tasks, called collaborative optimization. However, a study of collaborative optimization and how it applies to space transportation architecture analysis is beyond the scope of this research and shall be left for future work. It is important to note that the type of optimizers, as well as their implementation in an analysis framework, can have dramatic impacts on the analysis speed as there are many evaluations taking place throughout an architecture's subsystems to reach a single solution. Even though individual subsystem models may run quickly, depending on the level of nested optimizers, a high-level architecture optimization task may require lower-level subsystem models to be evaluated thousands of times for every high-level optimizer evaluation.

## *4.2 Ontology of Space Transportation Architectures*

Space systems architecting is a complex exercise in closing designs, converging on multiple vehicles and the elements of which they are composed. However, there is a hierarchical, structured order used to describe each vehicle, its elements, and what actions it takes. That is, there is an ontology that describes the architecture. Using the definitions discussed in Chapter 2.1 and applying them to space transportation architectures, the ontology's most basic terms and definitions, as well as their hierarchy, are defined as follows:

- **Campaign:** a unique combination of architectures assembled to achieve an overarching objective
    - **Architecture:** a unique pairing of a Vehicle to a Mission
        - **Vehicle:** a unique combination of Elements

  - · **Element:** a system composed of one or more subelements
    - + **Subelement:** a basic building block, representing a physical or functional decomposition
- ∗ **Mission:** a combination of CONOPs with Trajectories, manifesting as a unique sequence of events
  - · **CONOP:** a planned non-trajectory-related action or activity
  - · **Trajectory:** a physical path to be taken

When a vehicle is composed of elements, and is sized to a mission, an architecture is realized. Therefore, when multiple architectures are defined, a campaign is realized. Figure 36 represents this structure graphically.

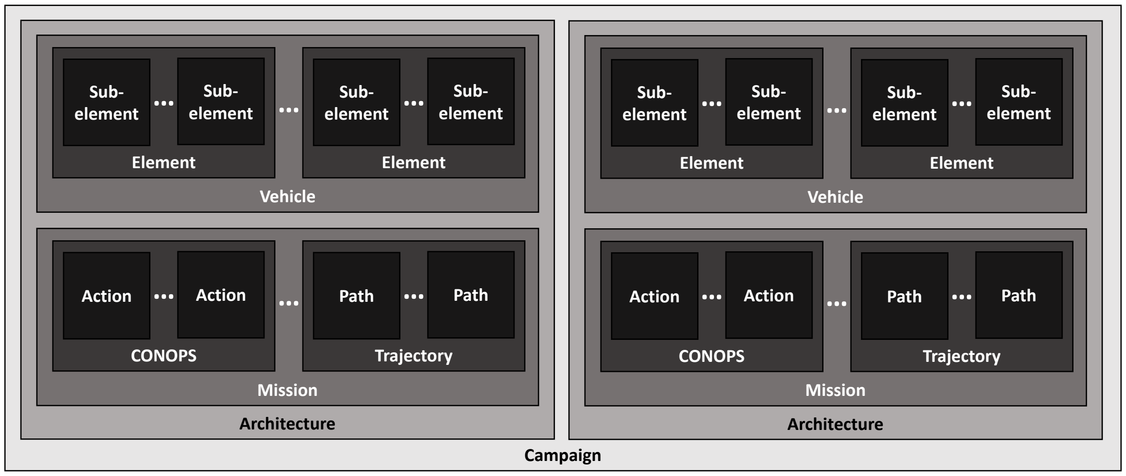


**Figure 36:** Graphical Space Transportation Architecture Ontology

### 4.2.1 Vehicle

Each element represents a building block of the vehicle. These elements are used to represent physical systems such as payloads and propulsive stages. These elements are in turn composed of subelement(s). At least one subelement is required to define the element, however, there is no limit to the number of subelements which may be used to define an element. Elements are sized based on various parameters. These sizing parameters may be parameters such as $\Delta V$, mission duration, number of crew, and operating environment of the mission, as well as parameters from other vehicle elements and subelements, including but not limited to power requirements, thermal loads, and volume.

### 4.2.2 Mission

To size an architecture correctly, the vehicle must be sized to a sequence of events as defined by the Mission. Any event of interest can be categorized as a CONOP or Trajectory-related event. A CONOP event is a change in inert mass ($\Delta m_i$) and/or a change in propellant mass ($\Delta m_p$). A Trajectory event is a change in velocity ($\Delta V$) and/or a passage of time ($\Delta t$). Table 8 provides a summary. It is important to note that CONOP events include both positive or negative changes, whereas Trajectory events are always positive.

**Table 8:** Event Type Decomposition

| Event Type | Event |
|---|---|
| CONOP | $\Delta m_i$, $\Delta m_p$ |
| Trajectory | $\Delta V$, $\Delta t$ |

This mission ontology allows architects to define any event of interest by using these event types in combination, e.g. docking with a propellant depot for refueling (a $\Delta m_p$), boiloff (a $\Delta m_p$ throughout a $\Delta t$), dropping a drop tank ($\Delta m_i$ with residual $\Delta m_p$), a burn ($\Delta V$ in a $\Delta t$), a coast event ($\Delta t$), etc.

### 4.2.3 Architecture

An architecture combines a unique vehicle and mission, where the vehicle and mission are defined as described above. The design of the architecture can be posed as an optimization problem. Consider the standard form for an optimization problem as

$$\begin{aligned} &min f(\vec{x}) && \text{objective function} \\ &subject\ to: && \\ &g_j(\vec{x}) \leq 0 && \text{inequality constraints} \\ &h_k(\vec{x}) = 0 && \text{equality constraints} \\ &x_i^L \leq x_i \leq x_i^U && \text{side constraints} \end{aligned}$$

As a simple example: $f(\vec{x}) = \sum_{i=1}^{n} m_{gross_i}$ where $f$ is the inert mass at LEO and $m_{gross_i}$ are the masses of the vehicle elements. $\vec{x}$ is the $\Delta V$ each propulsive element is responsible for, $g_j$ and $h_k$ define the minimum and maximum stage size constraints, and each $x_i^L$ and $x_i^U$ defines the upper and lower bounds, respectively, on each of the $\Delta V$s. Each element is composed of many subelements, which can be external codes or univariate or multivariate equations. These subelements are then sized according to $\vec{x}$ and any other inputs to the models. Through this process an entire system of systems space architecture is sized as an aggregate of its component subsystems.

## *4.3 Existing Tools*

In order to test the hypotheses discussed in the previous chapter, a modeling and simulation environment must be identified to act as a digital test bed for the purpose of evaluating architectures at the subsystem level while incorporating the effects due to technologies. Because the motivating field behind this research is in the space architecting domain, a digital test bed environment capable of evaluating space transportation architectures and technologies is desired. Table 9 provides a list of desirable features and their purpose in selecting a modeling and simulation environment to be utilized in a digital test bed for this research.

In a literature search for applicable tools to model space systems architectures, the following NASA tools were identified as potential candidates. Other tools may exist within the private sector, however, these tools are typically withheld as proprietary software and not made publicly available. The following is a brief description of each

**Table 9:** Required Features for a Modeling and Simulation Environment

| **Feature** | **Purpose** |
|---|---|
| Subsystem-Level Sizing Models | To enable the ability to evaluate entire architectures down to the subsystem level for the purpose of evaluating subsystem-level technologies |
| Ability to Integrate User-Provided Subsystem Models | To enable the use of user-trusted models to alleviate concerns regarding modeling technique employed by any one software package, as well as provide a flexible tool to evaluate a wide range of architectures which may not be considered initially |
| Ability to Analyze the Effects of Technologies | To enable the ability to evaluate the effects of infusing subsystem-level technologies at the architecture level of design |
| Integrated Vehicle and Trajectory Optimization | To enable evaluation of space transportation architectures as defined, consisting of both a mission and a vehicle |

of the identified NASA architecture sizing tool.

### 4.3.1 BLAST

Beyond LEO Architecture Sizing Tool (BLAST) is a tool developed by Zero Point Frontiers in cooperation with NASA's Johnson Space Center to in order to rapidly generate mass estimates for in-space transportation vehicles and architectures for human exploration missions [122]. The underlying mass estimating relations are historical data-based regressions ranging from the Apollo era up to space assets as of 2012, when the tool was released. Mass estimating relations are integrated into the tool to provide a user-friendly interface; however, this limits user visibility to the underlying regressions, as well as the ability to integrate new regressions. BLAST provides a platform for setting up and conducting trade studies and sensitivity analyses on the architecture.

### 4.3.2 COPA

The Computerized Orbital Performance Analysis (COPA) tool began development in the early 1990's as a FORTRAN-based architecture analysis tool. The tool was developed by NASA Marshall Space Flight Center's Advanced Concepts Office as a means to evaluate multiple architectures simultaneously. It has since been extended into a Microsoft Excel-based spreadsheet in the early 2000's and a Java extended application in 2012 to provide a simple user interface to the original FORTRAN code [71]. Vehicle elements are sized by element-level scaling equations of a fixed form with user-defined scaling parameters as follows:

$$W_{bo} = A + B * W_p + C * {W_p}^2 \tag{13}$$

where:

| | |
|---|---|
| $A =$ | weight of all components not dependent on propellant capacity |
| $B =$ | weight of components directly proportional to propellant capacity |
| $C =$ | weight of components that vary with the square of the propellant capacity |

COPA has the ability to track multiple separate vehicles simultaneously. These separate vehicles may be split and recombined in any fashion, where COPA manages tracking of which elements are on which vehicle. Propellant boiloff is calculated as either a fixed propellant mass per month or a percentage of the propellant load per month. Missions are defined by a fixed set of actions, namely, adding/dropping elements and specifying $\Delta V$ maneuvers subject to the ideal rocket equation:

$$\Delta V = g_0 * I_{sp} * \ln\left(\frac{m_0}{m_f}\right) \tag{14}$$

### 4.3.3 Envision

The Envision Exploration Vehicle System Estimation tool began development in 2001 to aid in quick-turnaround responses to mission and vehicle concept feasibility studies [26]. Envision is a Microsoft Excel-based tool developed by NASA's Johnson Space

Flight Center. It is composed of three primary layers: the main input layer, the system sizer layer, and the vehicle summary layer. Envision computes mass, volume, and power requirements for various subsystems, such as propulsion, structures, thermal protection, power generation, thermal control, life support, and avionics. Sizing of these subsystems is performed by either integrated physics-based models or internal mass estimating relationships based on historical data for each of the subsystems. The user is limited to the models provided in the Envision environment.

### 4.3.4 EXAMINE

The Exploration Architecture Model for In-space and Earth-to-orbit (EXAMINE) tool was developed by NASA's Langley Research Center. It aids in architecture definition and assessment prior to, or during, program formulation. It was developed in an effort to enable larger fractions of an architecture trade space to be assessed in a short time frame, while also allowing complex interactions between elements and systems to be quantitatively explored [60]. EXAMINE is a collection of Microsoft Excel workbooks utilizing the inherent features of Excel and Visual Basic for Applications to perform sizing and analysis. EXAMINE utilizes a collection of element parametric sizing models capable of sizing launch vehicles, hypersonic cruise and acceleration vehicles, in-space transfer stages, landers, entry vehicles, transfer habitats, orbital platforms, surface habitats, and other surface elements. New models are capable of being integrated due to EXAMINE's high level of modularity. Mission modeling is integrated into the EXAMINE tool and is capable of providing high-thrust and low-thrust trajectory estimations through its internal trajectory tool.

### 4.3.5 HExAM

The Human Exploration Architecture Model (HExAM) is a Microsoft Excel-based tool developed by NASA's Marshall Space Flight Center aimed at providing level-zero evaluation of various architecture options for manned exploration missions [91].

Vehicle elements are sized based on a mixed-form scaling equation with user-defined scaling parameters of the form:

$$m_{bo} = \frac{A}{1 + B/{m_p}^C} \tag{15}$$

This equation was found to fit the general scaling trends of in-space transportation stages very well. The scaling parameters $A$, $B$, and $C$ are determined by fitting data of historical designs. The mission events which size an element's propellant mass are subject to the ideal rocket equation of the form:

$$m_p = m_0 * \left[1 - exp\left(\frac{-\Delta V}{g_0 * I_{sp}}\right)\right] \tag{16}$$

Propellant boiloff is handled as a percentage of the total propellant lost per day. HExAM was later ported to the Python programing language utilizing a Qt-based user interface for quick and easy formation and manipulation of missions and vehicle element definitions. It also provides the ability to set up batches of cases where specified input parameters may be varied over user-defined ranges automatically.

## *4.4 The DYnamic Rocket EQuation Tool (DYREQT)*

An analysis tool must be selected to meet the research objective of this dissertation by conforming to the space transportation architecture ontology outlined in Section 4.2. The analysis tool needs to be capable of evaluating architectures at a subsystem level as defined by the ontology. This will require a collection of subsystem-level models to define architectures. Subsystem-level models are typically found independent of other subsystem models in literature. As such, the tool will need the ability to integrate various models developed outside of the tool itself. Additionally, the tool needs to have the ability to analyze the effects of technologies defined at the subsystem or component level, as described in Chapter 2. Finally, the ontology outlined shows that architectures integrate a mission containing a set of trajectories, and a physical vehicle composed of various elements, which must be analyzed and optimized together.

Within the industry, no single analysis tool or framework was identified which is agreed upon for the purpose of space transportation architecture design and analysis. Typically, each group or organization utilizes a different in-house developed tool with collections of models to perform architecture analysis. Each have their pros and cons; however, no tool meets the criteria of an analysis environment which meets the needed features discussed above. Table 10 summarizes the capabilities of the tools identified in this chapter against basic modeling and simulation environment requirements based on this research. BLAST has a simple and effective user interface. However, because of its compiled nature, integrating new subsystem models is not possible. This also limits the ability for a user to view the details of an underlying model and assumptions. COPA and HExAM both perform sizing at a system level utilizing various forms of equations to scale elements. This makes the sizing problem very simple and transparent, preventing any kind of subsystem level trades to be performed due to their very high-level nature. Envision and EXAMINE provide higher fidelity modeling on both the mission and vehicle sizing aspects of sizing an architecture. They are both modular Microsoft Excel tools, which allow for additional subsystem models to be integrated into the tools. However, this integration is not trivial and requires a substantial investment in time in order to properly integrate new models into the framework. These shortcomings prompt the formulation of a new space architecture design and analysis tool to meet the research objectives of this body of work.

At the Georgia Institute of Technology Aerospace Systems Design Lab, a team of researchers are working to develop a proof-of-concept MDAO environment for the design of space transportation architectures at the subsystem level. In its final state, the tool will be capable of integrating various user-provided surrogate models and physics-based tools together for sizing full architectures, while also providing fully

**Table 10:** Analysis Tool Comparison

| | | Tools | | | | |
|---|---|---|---|---|---|---|
| | | BLAST | COPA | Envision | EXAMINE | HExAM |
| Features | Subsystem Level Sizing Models | ✓ | ✗ | ✓ | ✓ | ✗ |
| | Ability to Integrate User-Provided Subsystem Models | ✗ | ✗ | ○ | ○ | ✗ |
| | Ability to Analyze Effect of Technologies | ✗ | ✗ | ○ | ○ | ✗ |
| | Integrated Vehicle and Trajectory Optimization | ✗ | ✗ | ✗ | ✗ | ✗ |

✓ = Met ○ = Partial ✗ = Unmet

integrated trajectory analysis and optimization concurrent with vehicle sizing. Simplicity is a primary focus, allowing users to integrate various models with ease. The purpose for this development is to introduce the ability to perform technology evaluations of large scale, complex, architectures in the space transportation domain, while providing integration and optimization of mission analysis and vehicle sizing unattainable with currently existing tools.

The Dynamic Rocket Equation Tool (DYREQT) is being developed to meet the goal outlined previously. This software will be utilized for performing experiments to test the research questions of this dissertation due to its unique capability to analyze architectures at the subsystem level with easy-to-integrate, user-provided subsystem level models used to define the vehicle, mission, and technology spaces. DYREQT is being developed in the Python programming language, utilizing modules such as SciPy and NASA Glenn Research Center's OpenMDAO to simplify the development of an MDAO engine for the framework [56, 37]. Together, these two modules allow much of the tedious work of connecting models to be automated, while also providing high levels of flexibility and modularity for modeling and optimization.

In its current state, as of mid-2017, DYREQT closely follows the ontology outlined in Figure 36. However, because DYREQT is incomplete, certain portions of the ontology are not fully realized, primarily, the trajectory portion of the mission. Rather, missions are purely a function of a series of events, which encompass both the action and path definitions of the basic ontology. However, the rest of the ontology is in place in the framework, with vehicles being a collection of elements, which are in turn a collection of subelements. Figure 37 shows the object dependence implemented in DYREQT.

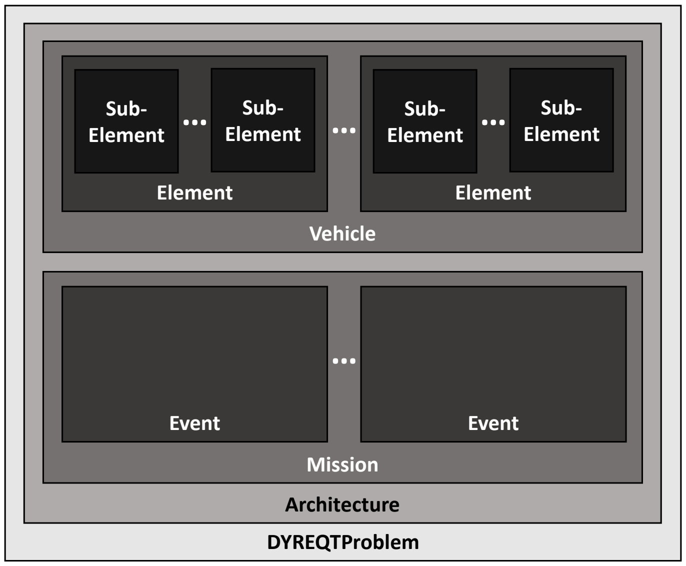


**Figure 37:** DYREQT Object Structure

Because DYREQT is built upon the MDAO framework, OpenMDAO, it is useful to understand the basic structure of some OpenMDAO concepts pertaining to modeling. OpenMDAO's primary structure consists of the Component, Group, and Problem classes. The Problem class is the top-level class for defining a root system to be solved. In OpenMDAO, the System class is the base class for the Group and Component classes. Groups are OpenMDAO Systems which contain other OpenMDAO Systems, either Groups or Components. Components are the most fundamental

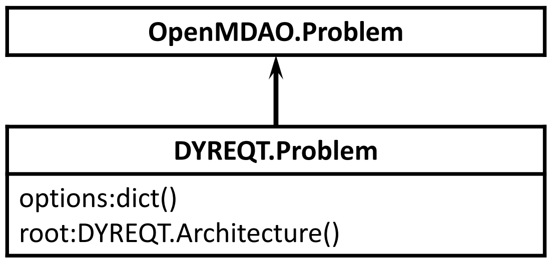


**Figure 38:** DYREQT Problem Class Structure

class, where basic models are written or wrapped. Components have sets of parameters and unknowns, where parameters are model inputs, and unknowns are model outputs. As mentioned previously, DYREQT allows for the automation of connecting models. This can be achieved by the use of basic naming conventions between models.

Classes in DYREQT are strongly related to the structure just described, as most DYREQT classes are subclasses of the OpenMDAO classes. The following sections are descriptions of the basic classes of DYREQT shown in Figure 37. This will help understand how DYREQT functions internally and differs from architecture sizing tools in existence. For a notional problem solved using DYREQT, see Appendix A for the setup and evaluation, and Appendix B for the output.

### 4.4.1 The DYREQT Problem Class

The DYREQT Problem class is a subclass to the OpenMDAO Problem class, as seen in Figure 38. This class provides the main entry point for creating and solving an MDAO problem. The main root system for the DYREQT Problem class is an instance of the DYREQT Architecture class. The Problem class also provides high-level options for control of file IO, console printing, optimizers, and numerical solvers.

### 4.4.2 The DYREQT Architecture Class

The DYREQT Architecture class is a subclass to the OpenMDAO Group class, which is in turn a subclass of the OpenMDAO System base class, as seen in Figure 39. Creation of an instance of this class provides the root system for the Problem class. The

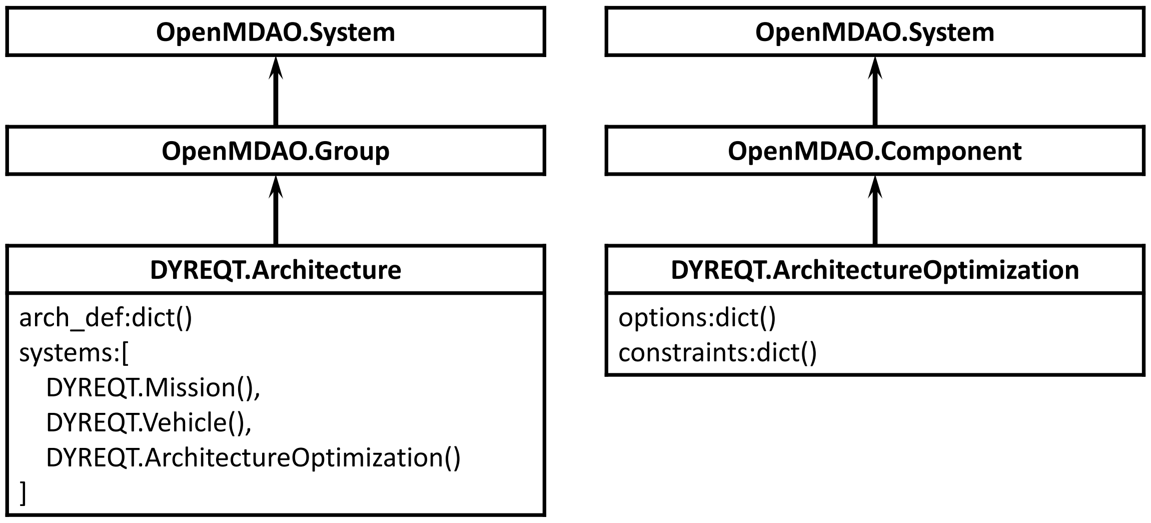


**Figure 39:** DYREQT Architecture Class and Helper Class Structures

DYREQT Architecture class contains three DYREQT class instances: DYREQT Mission, Vehicle, and ArchitectureOptimization objects. The Mission and Vehicle classes are described in detail in Section 4.4.3 and Section 4.4.5, respectively. The Architecture class takes three inputs: the mission definitions, the vehicle definition, and the CONOPs definition. Each of these inputs is a dictionary of key:value pairs defining each of the architecture components. These inputs are then parsed and used to feed the inputs to the DYREQT Mission and Vehicle classes. The Architecture-Optimization class provides architecture-level parameters such as the main objective value and high-level constraints on these parameters. Its class structure is seen in Figure 39.

### 4.4.3 The DYREQT Mission Class

The DYREQT Mission class is a subclass to the OpenMDAO Group class, which is in turn a subclass of the OpenMDAO System base class, as seen in Figure 40. The Mission class takes inputs provided by the Architecture class to create a set of DYREQT Event class instances and a MissionUtilities class instance. A detailed description of the DYREQT Event class can be found in Section 4.4.4. The MissionUtilities class provides mission-level parameters such as mission duration, as well as constraints on

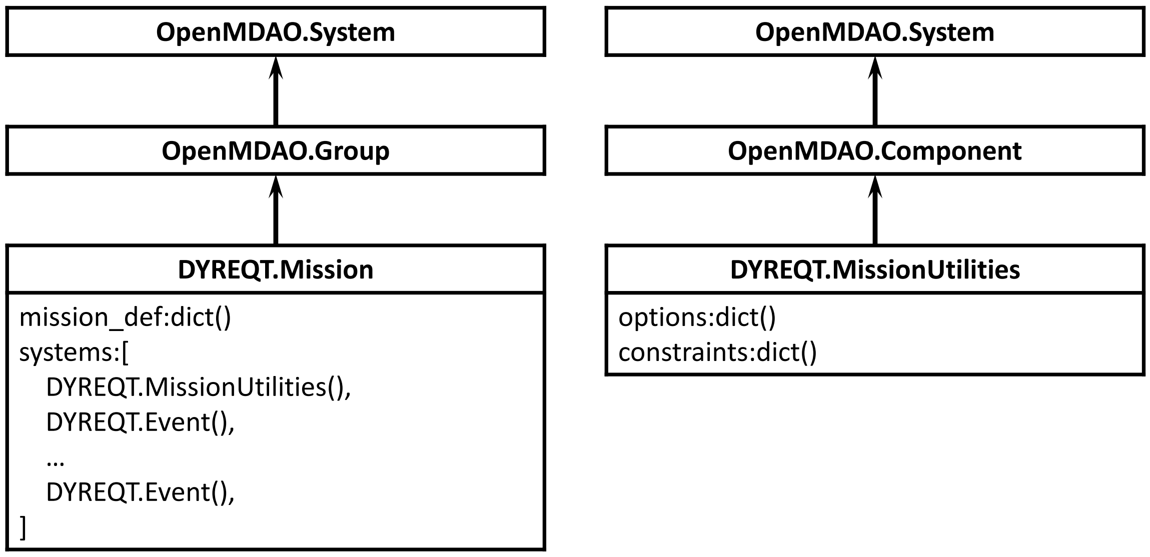


**Figure 40:** DYREQT Mission Class and Helper Class Structures

these parameters. Its class structure is seen in Figure 40.

### 4.4.4 The DYREQT Event Class

The DYREQT Event class is a subclass to the OpenMDAO Component class, which is in turn a subclass of the OpenMDAO System base class, as seen in Figure 41. The Event class is a base class for users to integrate external models with the DYREQT framework for the purpose of evaluating the defined mission. Inputs to these models flow from the original Architecture class inputs, through the Mission class, which parses the individual event inputs to the proper user model. The base Event class also allows inputs for constraints on event-level parameters. The DYREQT Event base class provides a collection of basic internal data, as well as basic model parameters, and unknowns shared with all events, regardless of type. The internal data includes a base name, total number of mission events, the current event number, and a list of active elements for the event, and parent mission object. The base parameters include vehicle gross mass and total payload. The base unknowns are initial and final mass at the beginning and end of the event, respectively. The user is able to link any other required model parameters and assign unknowns. If commonality exists

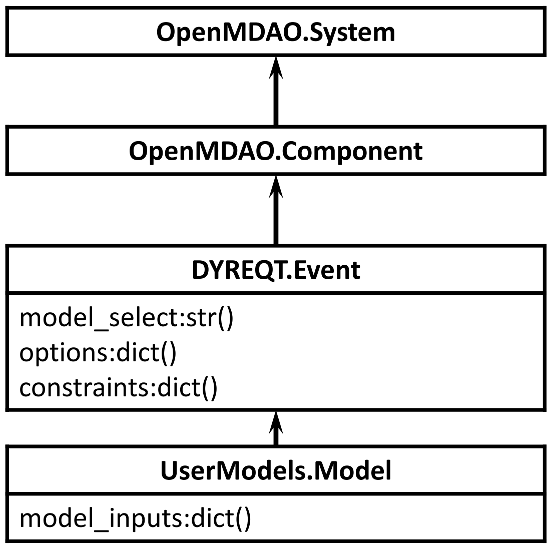


**Figure 41:** DYREQT Event Class Structure

between model parameters and unknowns anywhere in DYREQT, they will be linked automatically by DYREQT and will be available for other user models to interact with. There is no limit to the type of external model which may be integrated to perform mission-level calculations. Event models developed and integrated for the purpose of evaluating the experiments for this body of work can be found in Section 4.5.1.

### 4.4.5 The DYREQT Vehicle Class

The DYREQT Vehicle class is a subclass to the OpenMDAO Group class, which is in turn a subclass of the OpenMDAO System base class, as seen in Figure 42. The Vehicle class takes inputs provided by the Architecture class to create a set of DYREQT Element class instances and a VehicleUtilities class instance. A detailed description of the DYREQT Element class can be found in Section 4.4.6. The VehicleUtilities class provides vehicle-level parameters such as vehicle gross mass, as well as constraints on these parameters. Its class structure is seen in Figure 42.

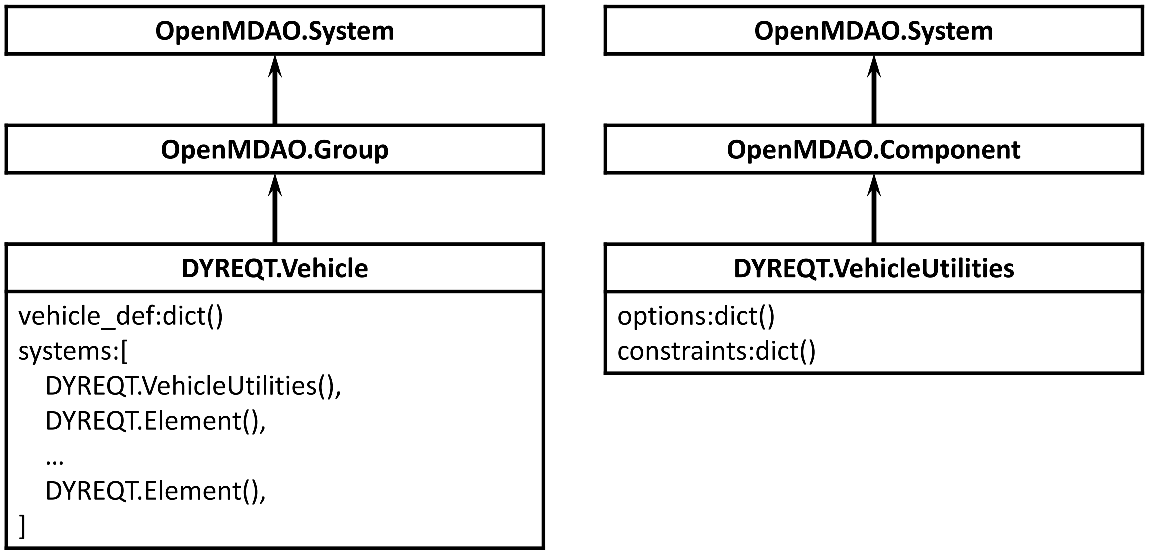


**Figure 42:** DYREQT Vehicle Class and Helper Class Structures

### 4.4.6 The DYREQT Element Class

The DYREQT Element class is a subclass to the OpenMDAO Group class, which is in turn a subclass of the OpenMDAO System base class, as seen in Figure 43. The Element class takes inputs provided by the Architecture class to create a set of DYREQT SubElement class instances and an ElementUtilities subclass instance. A detailed description of the DYREQT SubElement class can be found in Section 4.4.7. Here, the ElementUtilities class is a base class for two subclasses, the Stage and Payload classes. These utility classes were convenient to separate as they have dramatically different structures and requirements for sizing purposes. These two class structures can be seen in Figure 42. Though they have dramatically different structures and requirements for sizing, they are delineated by a single descriptor. If the element is sized by a propellant mass parameter, then it is a stage element utilizing the Stage utility subclass, otherwise, it is considered a payload element utilizing the Payload utility subclass, shown in Table 11. The ElementUtilities class is unique from the other helper class in that it provides unique inputs for the Stage and Payload subclasses, summarized in Table 12.

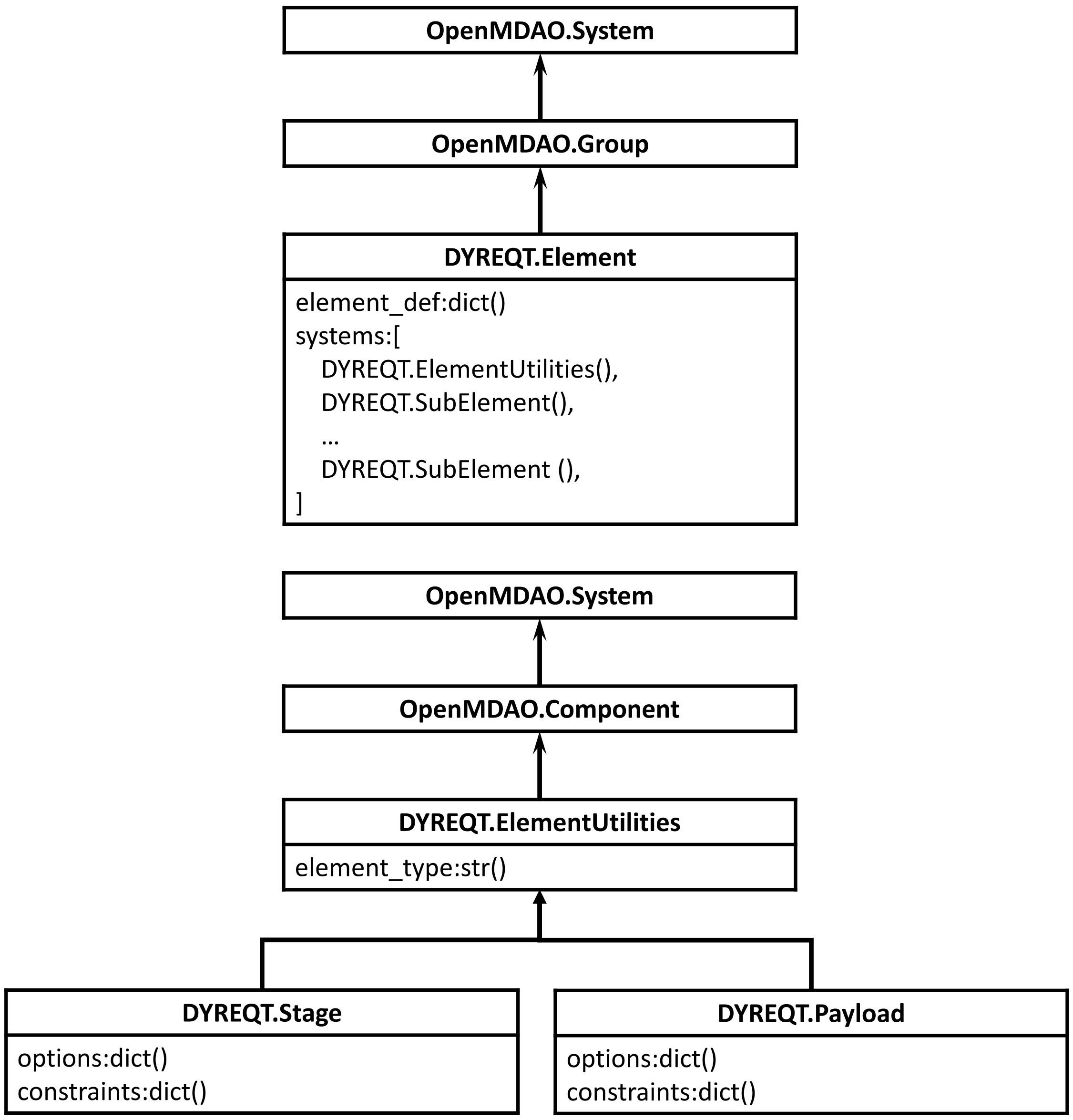


**Figure 43:** DYREQT Element Class and Helper Class Structure

**Table 11:** Element Type Decomposition

| Element Type | Sized by $m_{prop}$? |
|---|---|
| Stage | True |
| Payload | False |

**Table 12:** Element Class Inputs

| **Event** | **Input** | **Type** | **Range** | **Units** | **Description** |
|---|---|---|---|---|---|
| Stage/ Payload | auto_drop | bool | TRUE, FALSE | – | element is removed from the vehicle upon the completion of its terminal event |
| Stage/ Payload | mga | float | [0 : 100] | – | mass growth allowance as a percentage of dry mass |
| Stage/ Payload | pmr | float | [0 : 100] | – | program manager's reserve as a percentage of dry mass |
| Stage | mps_reserve | float | [0 : 100] | – | percentage of burned and boiled propellant to be added to the sized main propulsion system propellant load as a reserve |
| Stage | rcs_reserve | float | [0 : 100] | – | percentage of burned and boiled propellant to be added to the sized reaction control system propellant load as a reserve |
| Stage | boiloff_model | string | ['standard', 'HExAM', 'constant-rate'] | – | propellant boiloff model selector |

Stage elements are sized by the entire set of parameters available within DYREQT. Of primary importance is their ability to be sized by a propellant mass parameter, calculated from a total burn time for the element from the mission, coupled with thrust and specific impulse parameters of the element. These extra parameters and links to the mission greatly complicate sizing calculations compared to that of a payload. It also means that to define a stage element, one of its subelements must define a thrust and specific impulse parameter. Because of this extra complexity, it was desirable to separate stages and payloads to save computational effort during calculations. Payload elements are sized by the same set of parameters as stage elements, but with the absence of being linked to a mission change in velocity requirement. This means they will not require the unique set of parameters required by a stage, greatly simplifying their sizing.

### 4.4.7 The DYREQT SubElement Class

The DYREQT SubElement class is a subclass of the OpenMDAO Component class, which is in turn a subclass of the OpenMDAO System base class, as seen in Figure 44. The SubElement class is a base class for users to integrate external models with the DYREQT framework for the purpose of evaluating a defined vehicle element. Inputs to these models flow from the original Architecture class inputs, through the Element class, which parses the individual subelement inputs to the proper user model. The base SubElement class also allows for inputs for constraints on subelement level parameters. The DYREQT SubElement base class provides a collection of basic internal data and unknowns shared with all subelements, regardless of type. The internal data includes a base name, the subelement number, type, and parent element object. The unknown is an inert mass for the subelement. The user is able to link any other required model parameters and assign unknowns. If commonality exists between model parameters and unknowns anywhere in DYREQT, they will be linked

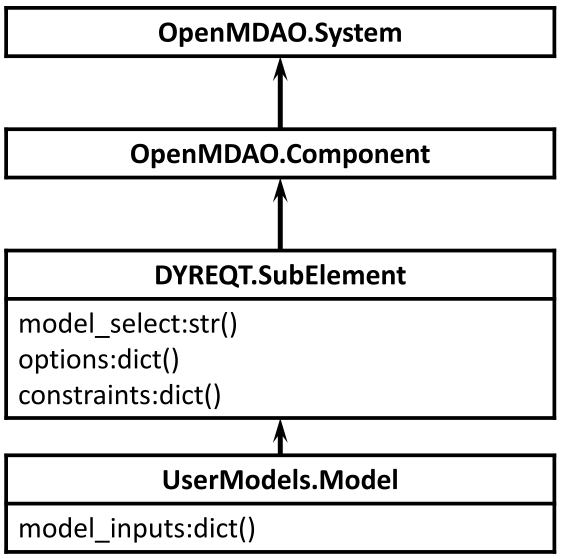


**Figure 44:** DYREQT SubElement Class Structure

automatically by DYREQT and will be available for other user models to interact with. There is no limit to the type of external model which may be integrated to perform vehicle element subsystem calculations. Vehicle subsystem models developed and integrated for the purpose of evaluating the experiments for this body of work can be found in Section 4.5.1.

## *4.5 In-Space Transportation Subsystem Modeling*

In order to perform the experiments set forth in this dissertation, a new analysis tool was needed, such that a digital test bed could be developed to perform experiments on the hypotheses. DYREQT was created to meet the need for an architecture analysis tool capable of integrating and evaluating user subsystem models. It integrates vehicle and mission optimization such that technologies and their effect on the high-level architecture space may be studied. However, because a tool of this nature did not exist within industry, the underlying models which need to be integrated into DYREQT for the purpose of evaluating technologies have not been developed in literature. Because of this, simple models which provide subsystem-level space element and mission sizing were developed which then could be integrated using DYREQT so that the primary research objective could be met. The following section details the models developed as DYREQT Event and SubElement subclasses. In addition to DYREQT, additional analysis is needed to provide metrics not provided by DYREQT, namely, an architecture cost metric and a distance metric between discrete architectures. Development of these analysis modules are documented in the following section as well.

### 4.5.1 Mission Models

DYREQT requires user models to be integrated for the purpose of mission analysis. This is done at the event level, as a subclass to the base DYREQT Event class. To model missions for this body of work, five mission event models were developed: burn, idle, mass delta, drop, and connect. These models are described below. For a table of available inputs to each model, refer to Table 30 in Appendix C.

#### *4.5.1.1 Burn*

The burn model is a subclass of the DYREQT Event base class. It is a model to represent changes in velocity of the vehicle. The model is developed around impulsive burn assumptions, utilizing the rocket equation. The concept of an “equivalent stage”, a functional representation of all active stage elements for the event, is utilized to determine a total burn time for the event. This calculated burn time is then passed, by DYREQT, to each of the active stage elements for the event. The purpose of generating a functional stage element, rather than directly using the physical stage elements, is because it allows the model to easily represent burn events with any number of active physical stage elements, also known as a parallel burn. In order to determine the total burn time of the functional equivalent stage, the following system of equations is employed:

$$t_b = \frac{m_p}{\dot{m}_{total}} \tag{17}$$

$$m_p = m_f * \left[ exp\left( \frac{\Delta V}{g_0 * I_{sp}} \right) - 1 \right] \tag{18}$$

$$I_{sp} = \frac{T_{total}}{g_0 * \dot{m}_{total}} \tag{19}$$

$$T_{total} = T_1 + T_2 + \cdots + T_n \tag{20}$$

$$\dot{m}_{total} = \frac{1}{g_0} \left( \frac{T_1}{I_{sp_1}} + \frac{T_2}{I_{sp_2}} + \cdots + \frac{T_n}{I_{sp_n}} \right) \tag{21}$$

The form of the rocket equation, Equation 18, is implemented because the problem is solved in reverse mission event order. A final mass is known, and an initial mass is calculated based upon the final mass and propellant mass. This allows for increased stability internally in the optimizers by limiting the occurrence of negative values during the iteration process which can result from poor initial conditions for mass estimates when solving in forward event order. The model is also capable of accounting for a flight performance reserve and attitude control maneuvers which may be required during the main burn. Both the flight performance reserve and attitude control factors are a correction to the input event $\Delta V$ parameters of the model. Appendix I.7 is the developed burn event model.

#### *4.5.1.2 Idle*

The idle model is a subclass of the DYREQT Event base class. It is a model to represent the flow of time during a mission. This allows time-based effects, such as propellant boiloff and crew consumables, to be modeled in vehicle elements. This is achieved by connecting the input $\Delta t$ to the $\Delta t$ parameter of the DYREQT Element class. Appendix I.10 is the developed idle event model.

#### *4.5.1.3 Mass Delta*

The mass delta model is a subclass of the DYREQT Event base class. It is a model to represent discrete mass changes to the vehicle during the mission. These discrete mass changes may be due to loading or offloading. This allows the modeling of discrete mass changes such as propellant reloading, garbage dumps, consumable resupply, and scientific payload loading. When acting as a propellant resupply event, the model allows for an automated resupply calculations where DYREQT will determine the proper amount of propellant required by the element for the mission without overloading it. This is done by connecting to the top_off input parameter of the DYREQT Stage class. The model is also capable of removing element subsystem masses during the mission through the use of a subelement index input for the specified active elements. Appendix I.11 is the developed mass delta event model.

#### *4.5.1.4 Drop*

The drop model is a subclass of the DYREQT Event base class. It is a model to allow the removal of full elements from the vehicle. Elements which are removed via this model are maintained in memory within DYREQT and can be reconnected to the vehicle. Appendix I.9 is the developed drop event model.

#### *4.5.1.5 Connect*

The connect model is a subclass of the DYREQT Event base class. It is a model to allow the addition of full elements to the vehicle. In order to connect an element, it must have been initialized during problem setup. This model does not allow the addition of entirely new elements not defined before problem initialization. For instance, to model a scenario where a vehicle attaches to a pre-deployed element, the element must be defined during the initial problem setup, then removed from the vehicle at the start of the mission via the Drop model. Appendix I.8 is the developed connect event model.

### 4.5.2 Vehicle Models

DYREQT requires user models to be integrated for the purpose of vehicle sizing. This is done at the subsystem level, as a subclass to the base DYREQT SubElement class. To model representative vehicles with the proper level of detail to allow technology impacts on the framework to be studied, six subsystem models were developed: avionics, engines, power, structures, tanks, and thermal. Each of the subsystems calculates an inert mass, power requirement, and thermal load that can be connected to other subsystem models to account for subsystem interdependence. The inert masses of all SubElement models in a DYREQT Element get added to become the total inert mass of that Element. These models are described below. For a table of available inputs to each model, refer to Table 31 in Appendix C.

#### *4.5.2.1 Avionics*

The avionics model provides sizing of hardware associated with sensing, actuating, and communication for the element. The model is derived from mass and power data of flight-certified, commercially available hardware [118]. The model allows the selection of a set of actuators, sensors, and communications packages from a predefined list to be used on the element. Actuators are scaled with element mass, while sensors are scaled by a user accuracy factor. The communications package is scaled based on the operational distance from earth. The model also allows the input of additional devices not defined by the model. The user must only specify a fixed mass and power requirement, and it will be added to the subsystem mass. Finally, the model accounts for wireless sensor technologies by applying a factor for cabling reduction to the overall mass of the subsystem. Total power requirement and heat load unknowns are defined for linking to other subsystem models. The developed avionics model is available in Appendix I.1.

$$m_i = \frac{m_{actuators} + m_{sensors} + m_{coms} + m_{other}}{f_{cable}} \tag{22}$$

$$P_{total} = P_{actuators} + P_{sensors} + P_{coms} + P_{other} \tag{23}$$

#### *4.5.2.2 Engines*

The engines model provides sizing of hardware associated with the physical engines of the propulsion system and associated plumbing. The model is capable of estimating the mass and power of a variety of engine classes: liquids, solids, nuclear, and electric. It is assumed that engines have negligible thermal loads on the spacecraft. Appendix I.2 contains the full engines subsystem model developed for this body of work. For liquid engines, the power required to drive the engine is not considered. The mass estimate is achieved via sets of scaling equations provided in Space Propulsion Analysis and Design, Section 5.3.1 [120]. These equations scale engine mass as a function of thrust as follows:

$$m_i = m_{engines} + m_{propmgt} + m_{misc} \tag{24}$$

$$m_{engines} = n_{engines} * \frac{T}{g_0}\left(\frac{T}{W}\right)^{-1} \tag{25}$$

where:

Monopropellants:

$$\frac{T}{W} = -3.7405 * 10^{-10}\left(T^4\right) + 7.1685 * 10^{-7}\left(T^3\right) - 5.221 * 10^{-4}\left(T^2\right) + 0.18761\,(T) - 0.039763 \tag{26}$$

Bipropellants:

$$\begin{cases} \dfrac{T}{W} = 6.098 * 10^{-4}\,(T) + 13.44 & \text{if } T < 50 \text{ kN} \quad (27) \\ \dfrac{T}{W} = 25.2 * \log(T) - 80.7 & \text{otherwise} \quad (28) \end{cases}$$

Similar to liquid engines, solid rocket motors assume no power requirement. The motor masses are modeled by a curve fit of historical data provided in Section 6.3 of Space Propulsion Analysis and Design [120]. The trends in the data show an increase in the propellant mass fraction with increasing propellant for smaller motors. Large motors have a tendency to have decreasing mass fractions with increased propellant loads due to large joints and thrust vector control hardware. This tipping point occurs at 10,000 kg of propellant. The system of equations for solid rocket motors is:

$$m_i = m_p * \left(\frac{1}{f_p} - 1\right) \tag{29}$$

Where:

$$\begin{cases} f_p = 0.0181 * \log(m_p) + 0.7962 & \text{if } m_p < 10{,}000 \text{ kg} \quad (30) \\ f_p = 0.0181 * \log(m_p) + 0.7962 & \text{otherwise} \quad (31) \end{cases}$$

Nuclear rocket engines are sized from level-zero physics equations found in Space Propulsion Analysis and Design [120]. The power required to operate the engine is assumed to be zero. The engine mass consists of the core and its related components. Tanks are not considered to be part of the engine. The model assumes an expander cycle for the turbopump assembly, with redundant turbopumps for reliability. The total inert mass of the nuclear engine is calculated as a sum of seven major components, shown by Equation 32.

$$m_i = m_{core} + m_{nozz} + m_{vessel} + m_{feed} + m_{cool} + m_{shield} + m_{tpa} \tag{32}$$

For details of the mass calculation for each of the seven components, refer to the nuclear method of the Engine class, found in Appendix I.2. Some of the components require detailed fluid calculations for gases at temperature and pressure. A fluids definition class and associated property calculations methods were developed primarily based on data from the National Institute of Standards and Technology Chemistry (NIST) WebBook [92]. Material properties for these definitions were obtained from the NIST WebBook as well as other sources [92, 108]. The developed fluids definition model can be found in Appendix I.12.

Electric engines are sized from a set of level-zero physics equations. The inert mass of the electric engine is calculated as the sum of 4 components, shown by Equation 33.

$$m_i = m_{thrusters} + m_{propmgt} + m_{pwrmgt} + m_{misc} \tag{33}$$

The user provides parameters such as, thruster specific mass, thruster efficiency, thruster power, thruster specific impulse, total thrust, and power management systems specific mass, used for sizing of an electric engine package. The total power requirement for the electric engines is calculated from the total number of engines required to meet the total thrust and thruster power inputs to the model. In addition to the physics-based equations for sizing thrusters, mass associated with propellant management and other miscellaneous hardware is estimated based on model fits of historical data [118].

#### *4.5.2.3 Power*

The power model sizes a spacecraft power generation system based on a total power required. The model has three primary components, the generator, power storage, and regulation/distribution. The model is capable of estimating the mass of two types of generators, either photovoltaic solar arrays or radioisotope thermoelectric generators (RTGs). Solar array mass is calculated from level-zero physics equations [118]. RTGs are scaled based on historical data [12]. The total mass and thermal load of the model are given by Equation 34 and Equation 35 respectively. Appendix I.3 contains the model developed for this dissertation and contains details about the sizing of the different components.

$$m_i = m_{gen} + m_{bat} + m_{reg} \tag{34}$$

$$Q_{total} = P_{req} * (1 - \eta_{tr}) \tag{35}$$

#### *4.5.2.4 Structures*

The structures model sizes primary spacecraft structural mass based on the design envelope area of the element. This model can be found in its entirety in Appendix I.4. This design envelope area is the surface area of the design volume of the spacecraft, shown notionally by Figure 45.

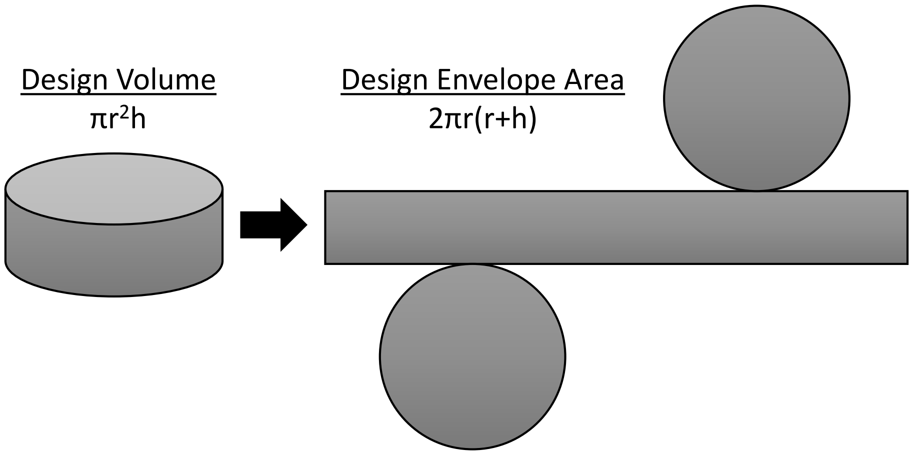


**Figure 45:** Notional Design Envelope Area

The design envelope area may be specified directly by the user, or left to the model to estimate based on a general form of the structure. The general form of the structure depends on the number of tanks specified by the design. If there are more than two tanks, the structure is assumed to be a disk. If there are exactly two tanks, the structure is assumed to be stacked. If there is only a single tank, the user specifies whether it has a truss or drop configuration. If none is specified, the model assumes a drop configuration. Figure 46 provides notional geometries for the basic structural configurations just described.

A cylinder is assumed to estimate the design envelope area for all configurations. For stacked, truss, and drop configurations, the radius of the structure cylinder is the radius of the largest tank, and the height of the structure is total height of the tank(s). To estimate the truss configuration, a density factor is applied to the cylinder to account for empty space in the truss. The disk configuration requires determining

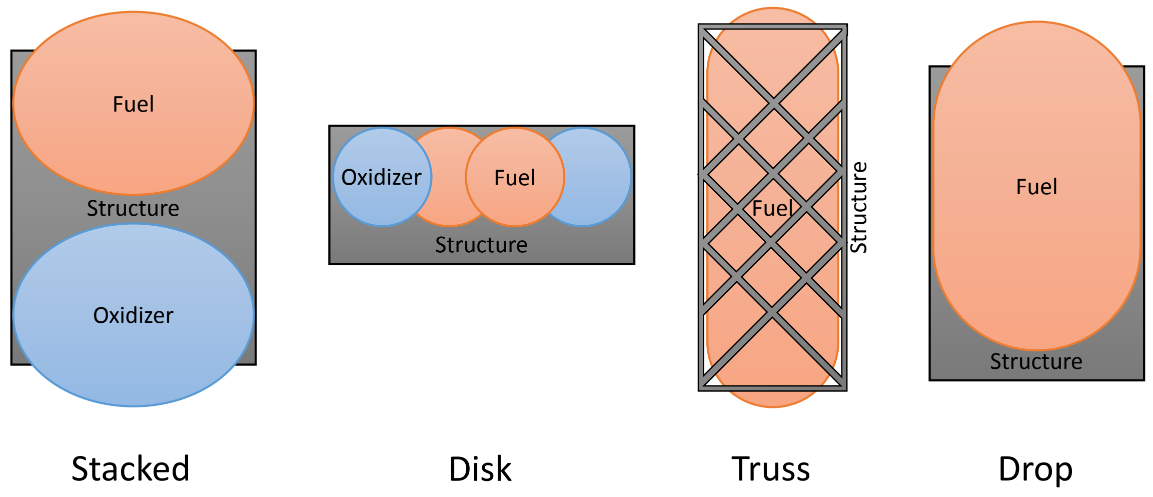


**Figure 46:** Notional Structure Configurations

the radius of a disk which will fit all the tanks. To do this, a weighted average tank radius is calculated based on the radius and number of each tank type. The concept of circle packing is utilized, where the radius of a circle which contains N equal circles within it becomes the design envelope radius. The ratio of the design envelope radius, $R_{de}$, over the average tank radius, $r_{avg}$, is related to the number of tanks, $n_{tanks}$, by Equation 36, a logarithmic fit of the data found in Kravitz's work on cylinder packing [61]. The height of the disk structure is a weighted average tank height.

Once the design envelope area is determined or provided, the inert mass of the structure is estimated based on the relationships given by Equation 37 [47], where $A_{de}$ is in ft$^3$ and $m_i$ is in lbm. The leading structure factor is determined based on the type of structure being estimated, shown in Table 13.

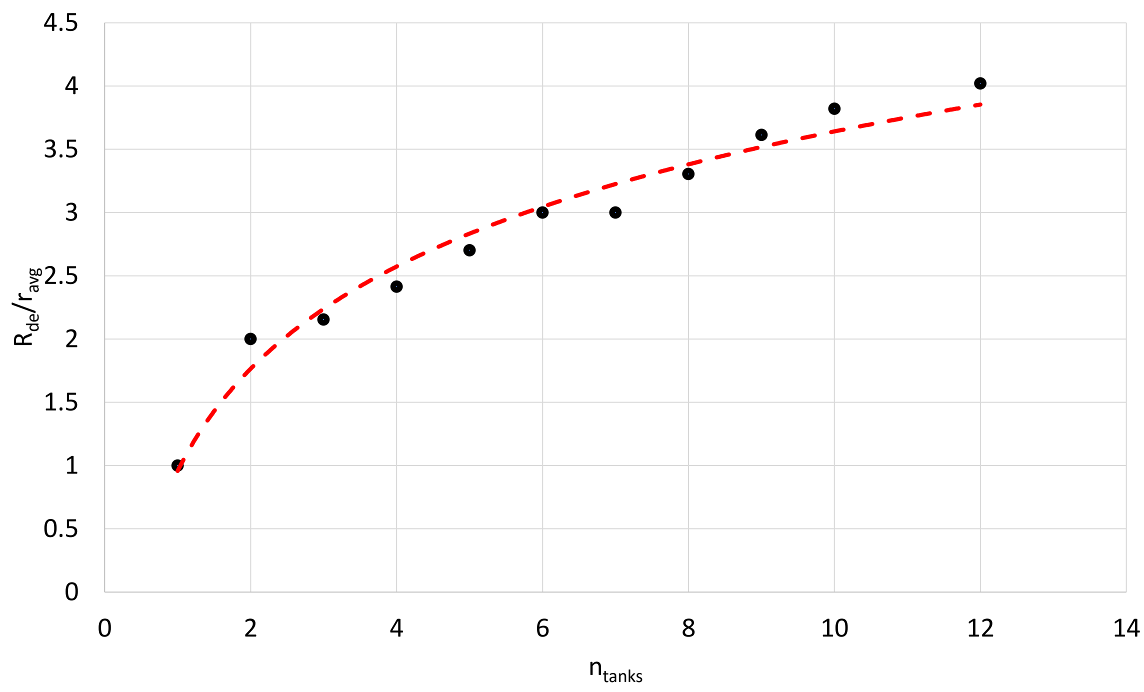


**Figure 47:** Ratio of Radii for Packing Circles [61]

$$R_{de}/r_{avg} = 1.1655 \ln\left(n_{tanks}\right) + 0.9571 \tag{36}$$

$$m_i = F_s * \left(A_{de}\right)^{1.15} \tag{37}$$

**Table 13:** Structure Factors

| Structure Type | $F_s$ |
|---|---|
| Disk/Stacked | 1.27 |
| Truss/Drop | 0.71 |
| Manned | 2.0 |
| Adapter | 0.99 |

#### *4.5.2.5 Tanks*

The tanks model sizes the propellant storage devices for the main propulsion system and reaction control system by Equation 38. This model uses inputs such as propellant mass, propellant types, propellant properties, pressurant type, number of tanks, and tank material properties to estimate the mass of all propellant storage devices. This is done using level-zero physics calculations for determining propellant volumes, and wall thickness of pressure vessels [16]. Assumptions are made for additional hardware such as inlet/outlet flanges, weld lands, brackets, and propellant/pressurant separation devices as seen in Equation 39. The model calculates the mass of pressurant required to expel all propellant from the tanks and accounts for isentropic expansion of a pressurized gas where necessary. Tank volumes are determined based on propellant properties estimated using the developed fluid definitions model in Appendix I.12 [92, 108]. The model is capable of assuming an integrated vehicle fluid management system where propellant between the reaction control system and main propulsion system is supplied via shared propellant storage tanks. This assumption removes the need for separate propellant tanks for the two propulsion systems, but adds additional mass to account for new hardware such as pumps and accumulators.

$$m_i = m_{propmgt} + m_{misc} + m_{pressurant} + m_{trap} + \sum_{i=1}^{n_{tanks}} m_{tank_i} \tag{38}$$

$$m_{tank} = m_{bare} + m_{weld} + m_{IO} + m_{sa} + m_{sep} \tag{39}$$

Sizing of the mass of the tanks assumes ideal pressure vessels of either oblate spheroid, sphere, or capsule shapes. The general shape of the tank is determined by a user-specified length over diameter ratio, shown in Figure 48. The tank's dimensions are then calculated based on this ratio and the required propellant volume by Equation 40 or Equation 41. A tank wall thickness is then calculated based on a user-supplied tank pressure and material properties by Equation 42. With a tank wall thickness and radius, an overall bare tank mass is calculated based on the material properties specified. Finally, additional mass is calculated for weld lands, inlets and outlets, structural attach points, and gas separation devices. The fully-developed tank model can be found in Appendix I.5.

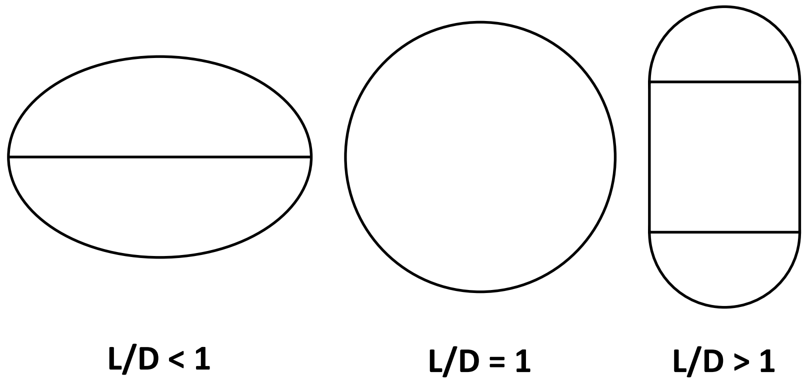


**Figure 48:** Basic Tank Geometries

$$\begin{cases} r = \left[\dfrac{6 * V}{\pi\left(3\frac{L}{D} - 1\right)}\right]^{1/3} & \text{if } L/D \leq 1 \quad (40) \\ r = \left[\dfrac{12 * V}{\pi\frac{L}{D}\left(3 + \left(\frac{L}{D}\right)^2\right)}\right]^{1/3} & \text{otherwise} \quad (41) \end{cases}$$

$$t = \frac{p_{tank} * SF * r}{2 * U} \tag{42}$$

### *4.5.2.6 Thermal*

The thermal model provides sizing of thermal control systems for a spacecraft. Calculations are separated into three parts, passive cooling, active cooling, and heat rejection. Appendix I.6 is the full thermal subsystem model developed through this dissertation. The total thermal control inert mass and power requirements are calculated by Equation 43 and Equation 44, respectivel.

$$m_i = m_{passive} + m_{active} + m_{rad} \tag{43}$$

$$P_{total} = P_{passive} + P_{active} \tag{44}$$

In the model, passive thermal control contains three primary components: multi-layer insulation, a liquid acquisition device, and a mass gauging device. The mass and power of passive thermal control are shown by Equation 45 and Equation 46, respectively. Multi layer insulation mass is a function of the number of insulation layers and either spacecraft geometry or tank geometry, depending if the tanks are internal or external to the primary structure of the spacecraft, respectively. The mass gauging and liquid acquisition device masses are estimates based on tank geometry [40, 19]. The mass gauging device is the only component of passive thermal control which requires power. The power required is a function of tank geometry [40].

$$m_{passive} = m_{mli} + m_{lad} + m_{gauging} = f\left(G, n_{layers}\right) \tag{45}$$

$$P_{passive} = P_{gauging} = f\left(G\right) \tag{46}$$

Active thermal control contains a collection of devices for rejecting excess heat in the propellant of the spacecraft to limit propellant loss through boiloff. The model assumes two primary components and associated hardware for this purpose, cryocoolers and broad area cooling shields, along with power controllers, circulating pumps, and tubing. The mass and power relations for the active thermal control components are shown by Equation 45 through Equation 46.

$$m_{active} = m_{cc} + m_{ctrl} + m_{circ} + m_{bac} + m_{tubing} = f\left(G, P_{active}\right) \tag{47}$$

$$P_{active} = P_{cc} + P_{circ} = f\left(G, p, Q\right) \tag{48}$$

Calculating the mass and power of the active thermal control components requires calculation of the heat which must be removed from the propellant to maintain zero boiled propellant. It is assumed that the amount of heat being deposited by internal sources is negligible compared to external radiation sources. To determine the external heat, the Lockheed Equation, Equation 49, is utilized to determine the amount of heat which passes through the multi layer insulation [30]. The cold side temperature is determined by spacecraft geometry and propellant vapor temperature. Propellant properties are determined by the developed fluid definitions model in Appendix I.12. The hot side temperature is calculated from the amount of heat being deposited from the external environment [73, 118].

$$Q_{mli} = A_{mli} * M * DF * \frac{1}{n_{layers}} * \left[C_s * \kappa\left(T_{avg}\right) * \overline{N}^{2.36}\left(T_H - T_C\right) + C_R * \epsilon_{mli}\left({T_H}^{4.67} - {T_C}^{4.67}\right)\right] \tag{49}$$

$$\kappa\left(T\right) = 0.017 + 7 * 10^{-6}\left(800 - T\right) + 0.0228 * \ln\left(T\right) \tag{50}$$

$$T_{avg} = \frac{T_H + T_C}{2} \tag{51}$$

The final major component estimated by the model is the mass of the thermal radiators used to reject excess heat to the environment, given by Equation 52. This requires estimating the area of the radiator, determined by Equation 53, which in turn requires calculating the total heat applied to the spacecraft from the external environment, along with any internal heat loads generated by the spacecraft [73, 63].

$$m_{rad} = A_{rad} * \rho_{rad} \tag{52}$$

$$A_{rad} = f\left(Q_{total}, \epsilon_{rad}\right) \tag{53}$$

### 4.5.3 Costing

Many of the assessments to be performed on the developed hypotheses require a multidimensional objective space. DYREQT and the developed mission and vehicle models, despite their advances in modeling space architectures, only provide a limited number of objectives, namely various masses of the architecture. It is desirable to generate cost metrics for the architectures such that there is a two-dimensional objective space of mass and cost. Many cost models exist in literature. This section will provide a brief overview of relevant cost models before selecting one to act as the model to provide cost data.

#### *4.5.3.1 NASA/Air Force Cost Model*

The NASA/Air Force Cost Model (NAFCOM) is a parametric estimating tool for space hardware [121]. The model utilizes historical data from NASA's Resource Data Storage and Retrieval Library and is primarily used during the early phases of development of projects. The model allows for the cost estimation at the subsystem or component level. Costing of the components follows the form shown by Equation 54. NAFCOM also allows process-based scheduling estimates, and time phasing of cost. The model provides a graphical user interface for inputting user information. NAFCOM does have the ability to output estimates to Excel spreadsheets where the original inputs may be manipulated for integration with external applications; however, if the basic form of the architecture is different, a new NAFCOM model must be manually set up and evaluated.

$$Cost = C * Weight^{W} * Inheritance^{X} * Technology^{Y} * Management^{Z} \tag{54}$$

#### *4.5.3.2 Project Cost Estimating Capability*

The Project Cost Estimating Capability (PCEC) cost model began development in 2013 to be a replacement to NAFCOM [4]. The underlying cost estimating relations (CERs) are derived from normalized data. Statistics about the underlying CERs within PCEC are publicly available; however, the CERs themselves, along with the underlying data are, unavailable to the public. Development of the initial set of underlying CERs was broken into two categories:

- Robotic Spacecraft
- Crewed and Space Transportation Systems

The robotic spacecraft CERs are multi-variable power equations developed using ordinary least squares regression of log-transformed data. The parameters selected were derived from principal component analysis of the original data set. The crewed and space transportation systems CERs are mostly single-variable regressions of mass. These two sets of CERs allow PCEC to evaluate cost estimates for systems such as earth-orbiting satellites, planetary probes, rovers, multi-stage rockets, liquid and solid engines, crew capsules, orbiters, and habitats. Currently, PCEC is not suited for estimating the cost of designs such as CubeSats, balloons, aircraft, nanosat launchers, or human hardware elements. Interfacing with PCEC is achieved via an Excel add-in and is heavily dependent on a user-in-the-loop to generate cost estimates. This type of interface is not well-suited for an automated and parametric design environment.

#### *4.5.3.3 Process-Based Economic Analysis Tool*

The Process-Based Economic Analysis Tool (P-BEAT) leverages complexity-driven CERs as opposed to mass-based CERs. Costs are estimated based on an activity build-up based on complexity of the component and the particular activity needed to transform the raw materials into a finished product [74]. The model is highly detailed, comprising over 50 development processes and 700 manufacturing processes [100]. The interface to P-BEAT is graphically-based, relying on a user in the loop to input data for all of these processes. Although providing a bottom-up approach to estimating a full life-cycle cost of a system based on how it is built could be advantageous, the level of information required to operate this model, along with its external interface, is not well-suited to the scope of this research.

#### *4.5.3.4 Software for Evaluating and Estimating Resources*

The Software for Evaluating and Estimating Resources (SEER) suite of tools was developed for estimating cost at a component level [33]. SEER estimates cost, scheduling, and reliability by comparing user entries with similar items in a historical database. The CERs are entirely obscured behind the tool. The user interface is highly complex and detailed, allowing modeling of nearly any system or component. However, this level of detail make it difficult to integrating SEER into an automated environment where a large variety of components are being estimated.

#### *4.5.3.5 TransCost*

TransCost is a system-level, historical mass-based cost estimation model for the cost estimation of space transportation vehicles [59]. The model uses a unique cost metric independent of annual currency changes. The form of the CERs in TransCost are single-variable power regressions of mass, similar to PCEC. TransCost breaks the cost of a vehicle into two components, the development cost and the fabrication cost. The total development cost, in man-years (MYr), takes the form shown by Equation 55. Each element and engine in the vehicle has a development effort, $H$ in man-years, associated with it, given by Equation 56. Each element is scaled by the system-specific constant, $a$, and system-specific cost-to-mass sensitivity, $X$. Other scaling factors, $f$, account for project systems engineering, technical development standards, technical quality, team experience, schedule, parallel contracting, and productivity.

$$C_D = f_0 * \left(\sum H\right) * f_6 * f_7 \tag{55}$$

$$H = a * m^X * f_1 * f_2 * f_3 * f_8 \tag{56}$$

The total fabrication cost, in man-years, takes the form shown by Equation 57. Each element and engine in the vehicle has a fabrication effort, $F$ in man-years, associated with it, given by Equation 58. Each element is scaled by the system-specific constant, $a$, and system-specific cost-to-mass sensitivity, $X$. Other scaling factors, $f$, account for project systems engineering and learning rate.

$$C_F = f_0 * \left(\sum F\right) \tag{57}$$

$$F = n * a * m^X * f_4 \tag{58}$$

### *4.5.3.6 Cost Analysis Module*

The developed cost analysis module is based on TransCost 7.1 and can be found in Appendix I.13. This model was selected due to its open nature, with all of its sizing relations available in the public domain [58], making integration with the level of data provided by DYREQT simple. It allows a custom module to be developed which utilizes a simple form of historical mass-based CERs which then can be operated in an automated fashion. The complexity, detail, and relative obscurity of the other models described become hindrances for the purposes of this research.

Inputs to the developed model are provided in Table 32 of Appendix C. The model is capable of estimating the development, production, and gross costs of the engines and vehicle subsystems of each stage of a vehicle, up to three stages. The model also estimates the total development, production, and gross cost of the entire vehicle. The cost of technology development is accounted for in the development standard factor, $f_1$. The engines and vehicle subsystem treat technologies independently. For instance, technologies applied to the structures subsystem will not affect the cost of the engines. However, the effects of utilizing multiple technologies are not accounted for in the implementation of the TransCost model utilized in this work. A fixed value for $f_1$ was utilized, regardless of number of technologies or the type of technology being considered. Also, though the TransCost model contains a cost growth factor for deviation from optimum time schedules, it was not considered in the implementation utilized by this work.

### 4.5.4 Architecture Similarity

Many of the research objectives of this dissertation require defining a metric which can be used to describe a level of similarity between a set of categorical options which define an architecture. With continuous and ordinal data, similarity is simple to define as a physical cardinal distance from one design to another. However, with categorical data, a physical distance cannot be directly interpreted, and hence a similarity becomes difficult to define. In 2008, Shyam Boriah performed a comparative evaluation of various techniques for defining a similarity measure between categorical data [11]. Each technique's effectiveness was determined by its ability to correctly identify outlier data points form the data set. Different techniques were well-suited for different data sets. Based on his conclusion, the Occurrence Frequency technique was selected due to its robustness in determining outlier data points across a wide range of data sets.

The principal concept of Occurrence Frequency is to evaluate a similarity between two discrete design points. The weighted sum of each category's similarity is the overall similarity of the two points, given by Equation 59. The weighting of each category can be defined in any manner; however, for simplicity, an even weighting for each category is used, given by Equation 60. For matching options within a category, a similarity of one is assigned, while mismatches are given a value less than one. Mismatches on less frequent options in a category within the data set are assigned a lower value than those on more frequent options. This relationship is given by Equation 61.

$$S\left(X,Y\right) = \sum_{k=1}^{d} w_k * S_k\left(X_k,Y_k\right) \tag{59}$$

$$w_k = \frac{1}{d} \tag{60}$$

$$S_k\left(X_k, Y_k\right) = \begin{cases} 1 & \text{if } X_k = Y_k \\ \left[1 + \log \frac{N}{f_k(X_k)} * \log \frac{N}{f_k(Y_k)}\right]^{-1} & \text{otherwise} \end{cases} \tag{61}$$

Multiple different design points can have the same similarity measure from a shared baseline. This is a result of the non-Euclidean nature of categorical data. There is no reason to say one option within a category is greater or less than another. Two different options, provided they have the same baseline option and frequency of occurrence, will result in the same similarity value for that category, despite being two different options. This phenomenon will make it difficult with large architecture spaces to evaluate the similarity between any two architectures. One could theoretically calculate a relative similarity to every other design point in a data set and compose that information into an overall similarity of the design point to the design set, but with the number of alternatives being considered by this research, the number of evaluations becomes unmanageable. For example, if a design set contains 100 designs, each of the 100 points must be evaluated against the other 99 design points. This results in a total of 9900 similarity evaluations, scaling roughly as $N^2$, where $N$ is the number of design points. For this research, a single, randomly selected, design point from the data set acts as a common baseline to all architectures for the purpose of estimating architecture similarity.

## *4.6 Model Validation*

The development of DYREQT and the subsystem models throughout this chapter provide the ability to create a digital test bed on which the hypotheses of this dissertation may be tested. The development of DYREQT and these models represents a significant contribution to the space architecture community. The digital test bed environment has suffered from the lack of utilization of industry standard tools. Proving a level of validity of these models and tools is critical in establishing the analogs nature of these new tools and models to accepted industry data.

The mission models are simple, only relying on manipulations of the rocket equation, mass addition, mass subtraction, and time additions. It was determined that these models do not require detailed validation against literature data, but rather verification through simple use cases. To do this, simplified vehicle models were employed to mimic the functionality of HExAM. Identical missions were evaluated with both tools to determine validity of the mission models developed. Results between both tools were within 0.02% across mission events. Differences are likely due to rounding errors between the two tools. Validation results can be found in Appendix H.

For subsystem models which employed regressions from literature, outputs were verified to match those of the original regression from their respective literature sources. However, due to a lack of detailed mass breakdown data of stage elements in literature, level-zero physics-based subsystem models were difficult to validate independently. Instead, validation of the underlying subsystem models is implied by using the collection of developed models to estimate the mass of a variety of stage types spanning the capability of the subsystem models. DYREQT was utilized to integrate the various models discussed in this chapter. Differences in the burnout mass of the vehicle elements were within 5% of the validation designs in most cases. In some extreme cases, differences in estimation were as high as 25%; however, variations in the mass estimates from literature data are explained by observation of assumptions made by the underlying subsystem models. For instance, the Centaur upper stage is a particularly structurally mass efficient design for the type of stage. The models developed do not estimate outlier designs. However, the model can be forced to estimate the Centaur structural mass, which then brings the overall error in inert mass estimation to within 5%. The other validation point that was estimated with a high difference was the methane cryogenic propulsion stage. This can be explained by the uncharacteristically high structural mass of the reference design. However, the models are accurate for performing conceptual design across many different architecture concepts while capturing general trends due to these architecture choices. The full set of validation results can be found in Appendix H.

# CHAPTER V

# EXPERIMENTATION & IMPLEMENTATION

In general, experimentation can be thought of as having three main phases, described as follows:

1. **Thought Experiment:** Initial concepts and ideas are explored through the use of simple logic and small notional problems aimed at providing evidence in support of further investigation through formal experimentation. This process is typically initialized via an exhaustive literature search.

2. **Experimental Design:** Once a research question has been deemed worthy of further investigation through thought experiments, formal experiments must be developed to test the hypothesis developed. This typically requires physically developing models and setting up physical experiments.

3. **Design of Experiments:** After it is known what the experiment will consist of, a logical set of inputs must be selected in order to observe and obtain information such that the research question and hypothesis can be answered.

Throughout Chapter 3, thought experiments were described which aided in developing research questions and hypotheses which make up the body of work of this dissertation. To further examine these hypotheses, rigorous testing shall be performed. In order to perform this testing, experiments will be developed such that a design of experiments may be performed to study the research questions. The following subsections are a description of the experimental design, design of experiments, and results to each hypothesis in Section 3.4.

## *5.1 Experimentation*

Utilizing the models developed in Chapter 4.5, combined with the DYREQT tool developed in Chapter 4.4, the scope of the trade space available for experimentation is very large. The approximately 105 inputs to the various subsystem models and DYREQT were mapped to 45 high-level architecture parameter and 10 technologies. These architecture and technology space options were selected based on the available design space inputs derived from the developed subsystem models discussed in Chapter 4.5. This original full factorial DOE resulted in a total of $4.810 * 10^{21}$ compatible alternatives, far too many to evaluate. This is a result of the combinatorial problem discussed in Chapter 3.3.2. To reiterate, the problem of combinatorial explosion is outside of the scope of this research. As such, the conjecture to Research Question 4 states that subsets of the architecture space will be selected to minimize combinatorial explosion such that the main objective of this body of work may be performed.

### 5.1.1 Digital Test Bed

The architecture trade space was reduced to contain approximately 30 categories. Down-selection focused on maintaining vehicle options while reducing mission options from the architecture space due to the simple mission event models developed through this body of work. Because the vehicle options account for most of the design variability, the simplest way to reduce the number of alternatives was to reduce the number of independent stage elements. Each stage element is defined by 10 parameters, and each individual element may be paired with any other element in a multi-stage vehicle. Due to the relatively simple mission modeling developed for this dissertation, a small mission subset was chosen. Seven technologies were selected, resulting in nine technology combinations: each-one-on(7), all-on(1), all-off(1). These sets were selected to further reduce the total number of alternatives being evaluated. The reduced architecture space contains a total of 8,946,432 architectures, fully within the capability of DYREQT. The final vehicle space utilized as a starting point for experimentation is represented by Table 14. The mission space considered for all experiments consists of the options listed in Table 15. Finally, the technology space and its options is represented by Table 16. The collection of the options in these tables constitute the entire architecture and technology spaces considered throughout the experimentation of this dissertation. Each experiment further narrows the scope of these spaces to focus the resulting data such that specific observations may be made.

In order to obtain meaningful data for the purpose of observation and analysis, objective metrics identified through the hypotheses must be evaluated. Table 17 provides a summary of the objective space metrics required by each of the experiments, identified by a mark in the respective cells. The selection of these objective metrics will be examined in further detail in the sections that follow. For the current discussion, it is sufficient to know that these are the objective metrics which must be evaluated for each of the architecture alternatives.

Figure 49 provides the basic structure of the digital test bed developed for this dissertation. The architecture and technology spaces, together, feed the inputs to the design space, which contains the modeling and simulation environment. The modeling and simulation environment identified consists of DYREQT and the subsystem models discussed in Chapter 4.4 and Chapter 4.5, respectively. These provide the capability to integrate architecture sizing and technology evaluation at the subsystem level in the space transportation domain.

The final step to allow the examination of the hypotheses of this body of work is to establish the connections between the various system spaces of the digital test bed. The architecture and technology space options enumerated in Table 14 through Table 16 must be mapped to the design space attributes, defined by the modeling and simulation environment. Additionally, outputs from the modeling and simulation environment must be mapped to the objective space metrics listed in Table 17. These

**Table 14:** Architecture Categories Derived from the Vehicle Trade Space

| **Category** | **Options** | | | | | | |
|---|---|---|---|---|---|---|---|
| **Vehicle** | | | | | | | |
| Number of Stages | 1 | 2 | | | | | |
| Payload Mass(kg) | 1000 | 10000 | | | | | |
| **Element$_{\mathbf{n}}$** (Repeat for Each Stage) | | | | | | | |
| MPS Class | Liquid | Solid | Nuclear | Electric | | | |
| MPS Propellant | $LOX/LH_2$ | $LOX/LCH_4$ | NTO/MMH | Xenon | $LH_2$ | $N_2/H_4$ | Solid |
| RCS Class | Liquid | | | | | | |
| RCS Propellant | NTO/MMH | $N_2/H_4$ | | | | | |
| Pressurant | Helium | | | | | | |
| Tank Configuration | Stacked | Disk | Single | | | | |
| Structure Type | Manned | Unmanned | | | | | |
| Power System | Solar | RTG | | | | | |
| MLI Layers | 20 | 60 | | | | | |
| Communication Type | Near Earth | Deep Space | | | | | |

**Table 15:** Architecture Categories Derived from the Mission Trade Space

| **Category** | **Options** | |
|---|---|---|
| Destination Duration | Long | Short |
| Inbound Correction Maneuver | Small | Large |

**Table 16:** Technology Categories Derived from the Technology Trade Space

| **Category** | **Options** | |
|---|---|---|
| Wireless Sensors | TRUE | FALSE |
| Low Leak Valves | TRUE | FALSE |
| High Capacity Energy Storage | TRUE | FALSE |
| Composite Structures | TRUE | FALSE |
| Composite Propellant Tanks | TRUE | FALSE |
| Integrated MPS/RCS Propellant Storage | TRUE | FALSE |
| Active Cryocooling | TRUE | FALSE |

**Table 17:** Experimentation Objective Metrics

| | **Experiment** | | | |
|---|---|---|---|---|
| **Objective Metric** | **1** | **2** | **3** | **4** |
| Vehicle Gross Mass (kg) | × | × | × | × |
| Vehicle Gross Cost (MYr) | × | × | × | × |
| Similarity | | × | × | |
| Vehicle PMF | | | × | × |
| Stage Boiloff Rate (kg/day) | | | × | |

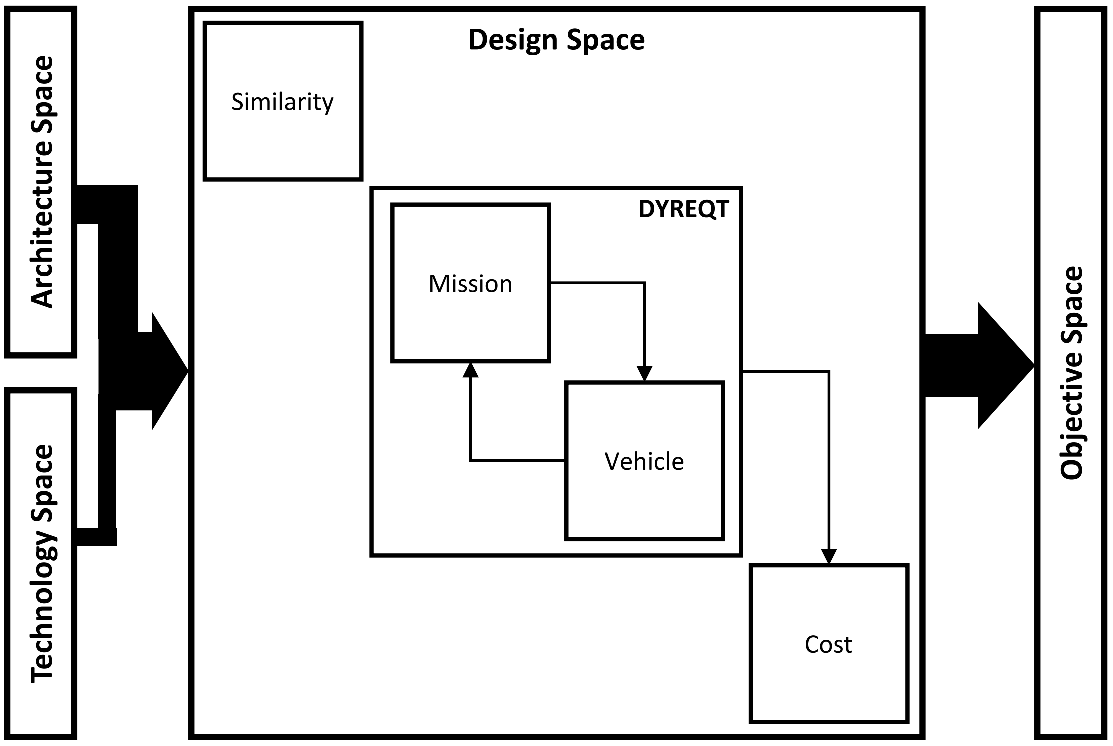


**Figure 49:** Digital Test Bed for Experimentation

mappings of the system spaces follow the technique described in Chapter 3.3.1.2. Table 18 provides the mapping of the architecture space parameters to the design space attributes. The various options of the architecture parameters specify values of these attributes within the design space. The specific values of the design space attributes based on the architecture space parameter options can be found in Table 33 through Table 45 of Appendix C. Similarly, Table 19 provides the mapping of the technology space parameters to the design space attributes. The various options of the technology parameters specify values of these attributes within the design space. The specific values of the design space attributes based on the technology space parameter options can be found in Table 46 through Table 53 of Appendix C.

The mapping of the design space to the objective space is straightforward in this case. The mappings of the technology options simply connect directly to the activation flags within the modeling and simulation environment. This is because the

**Table 18:** Architecture Space Parameter To Design Space Attribute Mappings

| **Architecture Space Parameter** | **Design Space Attributes(s)** |
|---|---|
| **Vehicle** | |
| Number of Stages | event_list, element_list |
| Payload Mass | mass |
| **Stage(s)** | |
| MPS Class | start_penalty_mps, total_thrust_mps, engine_thrust_mps |
| MPS Propellant | isp_mps, mixture_ratio_mps |
| RCS Class | start_penalty_rcs, total_thrust_rcs, engine_thrust_rcs |
| RCS Propellant | isp_rcs, mixture_ratio_rcs |
| Pressurant | pressurant |
| Tank Configuration | num_fuel_tanks_mps, num_ox_tanks_mps |
| Structures Type | manned |
| Power System | generator_type |
| MLI Layers | mli_layers_mps, mli_layers_rcs |
| Communication Type | comms_type |

**Table 19:** Technology Space Parameter To Design Space Attribute Mappings

| **Technology Space Parameter** | **Design Space Attributes(s)** |
|---|---|
| Wireless Sensors | wireless_sensors |
| Low Leak Valves | start_penalty_mps, start_penalty_rcs |
| High Capacity Energy Storage | storage_specific_energy |
| Composite Structures | composite |
| Composite Propellant Tanks | composite_fuel_tanks_mps, composite_ox_tanks_mps, composite_fuel_tanks_rcs, composite_ox_tanks_rcs |
| Integrated MPS/RCS Propellant Storage | ivfm |
| Active Cryocooling | active_cooling_mps, active_cooling_rcs |

modeling and simulation environment selected provides analysis of technologies within the various subsystem models. The effects of technologies are calculated directly within the subsystem model, which then are propagated to the objective metrics. This is in contrast to the more traditional technique of utilizing K-factors on specific inputs and outputs of the analysis to account for technology impacts. The performance of the various technologies is incorporated into the subsystem models.

The digital test bed described above will be utilized by each of the experiments to follow. Each experiment contains different subsets of the architecture, technology, and objective spaces described above. However, the mapping of the architecture and technology parameters to the design attributes, and from the design space outputs to the objective space metrics is consistent. For each of the experiments, any post-processing in the form of filtering and data exploration was performed using SAS® JMP® software package, for its ability to handle large data sets.

### 5.1.2 Experiment 1: Performing Technology Evaluation Before Design Down-Selection

**Research Question 3**

Is the paradigm of down-selecting to a baseline design on which to perform technology analysis sufficient for the exploration of complex architectures?

The purpose of experiment 1 is to determine if the traditional paradigm for technology evaluation holds for the system of systems problem. Traditional, technology evaluation calls for design down-selection and optimization before actually analyzing technologies. However, this paradigm assumes that the overarching design of interest is a system which technologies are applied to, as opposed to a system of systems. A notional example was examined in Section 3.3.2 providing good evidence for the hypothesis repeated below:

**Hypothesis 3**

The paradigm of down-selecting to a baseline design and then performing technology analysis will not be sufficient in performing architecture design. This paradigm assumes that the systems a technology acts upon remain constant throughout the down-selection process. Because these systems vary between architectures, the effects of technologies will be inconsistent among these architectures.

##### *5.1.2.1 Procedure*

The first part of this experiment shall be to test the four alternatives from the notional example in Section 3.3.2, utilizing the larger, more complex models developed to observe whether the assumptions made during the notional example are indeed observable. This is important because the notional example makes many unrealistic assumptions, namely, the interdependence of subsystems in the vehicle. To test this, similar vehicles to those discussed in the notional example will be set up and evaluated using DYREQT and the model developed in Chapter 4.5. This will validate the notion that down-selection can indeed limit the best observable architecture.

Once this has been established, fully testing the hypothesis requires observing the composition of optimal architectures in the resulting objective space. Because the objective space is two-dimensional, consisting of architecture mass and architecture cost, there will exist a two-dimensional Pareto front of designs which are optimal. The architecture space consists of all compatible single-stage options from Table 14 and Table 15. The technology spaces to be tested in this experiment will focus on the all-off, all-on, composite propellant tanks, and active cryocooling technology sets. Table 33 through Table 53 in Appendix C provide the specific design space attribute values associated with the options for each of the architecture and technology space options.

Appendix D provides the default set of values for unmapped design space attributes within the modeling and simulation environment. The observation will be the number of architectures with a given main propulsion system propellant type in the set of Pareto optimal architectures in the objective space. A shift in the distribution of architectures with certain propellant types in the Pareto optimal set will be strong evidence in support of Hypothesis 3, as this will indicate that technologies indeed have an impact on the optimal architectures and should be considered before down-selection.

The trade space considered will focus on single stage architectures, limiting the propellant types to liquid bipropellants, nuclear, or electric. The reason for removing solid rocket propellants is because they tend to dominate the Pareto optimal set of the objective space due to their relative simplicity and lower mass, coupled with lower cost. This is not to say that there is an issue with the model, but rather, for the narrow mission space selected, solids tend to dominate. Because the hypothesis deals with observing shifts in the composition of architectures in the Pareto front, solids were removed from the trade space to allow a more competitive mix of alternatives to be studied based on the missions being tested.

#### *5.1.2.2 Results*

The architecture space evaluated consists of a total of 18,432 architectures. Figure 50 shows the distributions of architectures by the main propulsion system propellant type among the entire objective space. The $LH_2$ and Xenon propellants have fewer architectures due to compatibilities in the architecture space which results in lower numbers of alternatives for those specific architecture types. Figure 51 is the objective space for experiment 1. The overall architecture mass is limited to 500,000 kg for visualization purposes. One will notice that there are no Xenon-based architectures in the objective space when limited to this mass. This is due to poor assumptions in the default vehicle which requires power to the electric engines to be fully supplied by batteries in the power subsystem during an eclipse cycle, grossly oversizing the battery mass. In a real design, there would be mission constraints limiting electric engine operation during eclipses to limit this interaction.

The four case notional example presented in Chapter 4.5 was evaluated using DYREQT and the developed subsystem models. The results from the original example are reprinted, along with the results from DYREQT, shown in Table 20. Overall, the performance of a specific architecture was lower when calculated by DYREQT compared to the notional calculations. This is not an error in DYREQT, but rather a result of its ability to account for subsystem interactions. Where the notional calculations assume no subsystem interaction, DYREQT is able to account for interactions such as growth in the propellant tanks due to varying propellant loads, which in turn affects the structural mass, or growth in the power subsystem to account for the increased power requirements imposed by the technology-enhanced thermal control subsystem. The ability to account for these interactions while introducing technologies at the subsystem level was the initial motivation behind the development of DYREQT. However, despite these kinds of interactions being accounted for in DYREQT, a similar trend described by the notional example is observed in the results from DYREQT. In both cases, without technology, the storable architecture had the lowest of the two inert masses. Under the traditional paradigm, the storable architecture would be selected to move forward with technology evaluation. However, in both cases, the technology-enhanced methane architecture performs best, indicated by the lowest inert mass of any alternative. This architecture would be overlooked under the traditional paradigm of architecture down-selection before technology evaluation.

This notional example is just one example of a shift in the compositions of architectures in the objective space due to technologies. However, this observation is

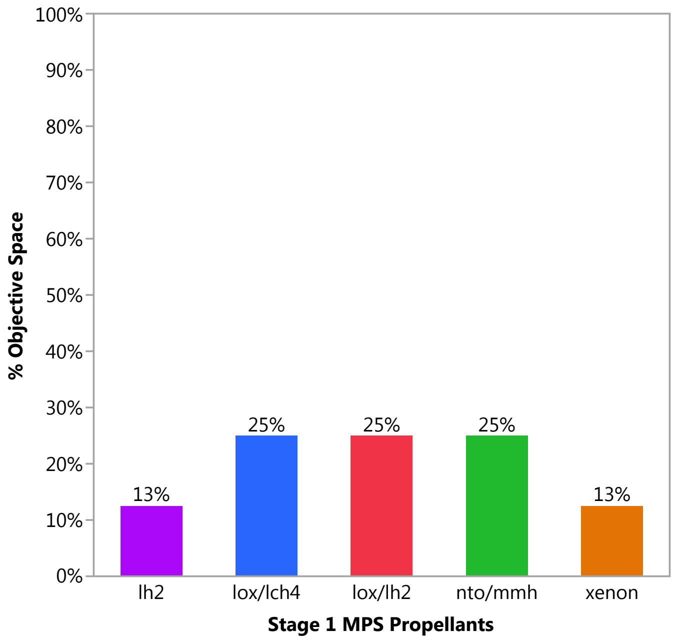


**Figure 50:** Experiment 1 Objective Composition

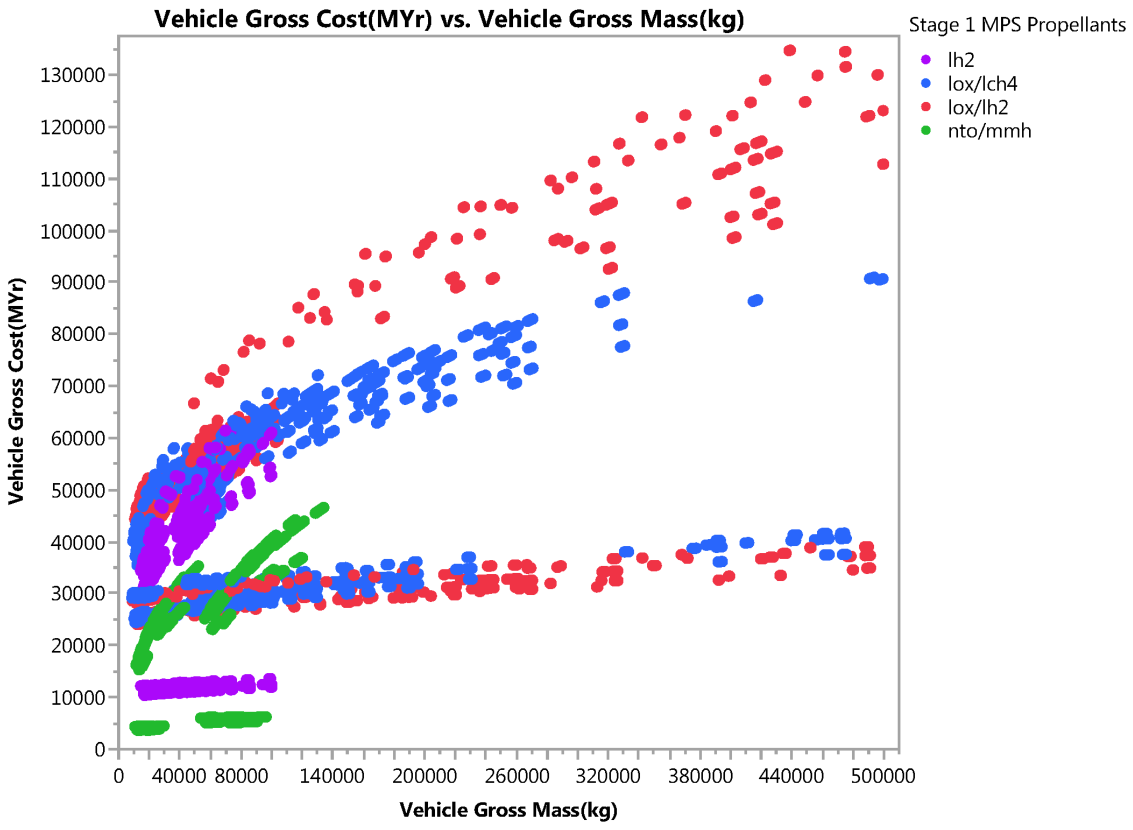


**Figure 51:** Experiment 1 Objective Space

Table 20: Notional Example Results Compared to DYREQT Results

| **Architecture** | **Notional Inert Mass(kg)** | **DYREQT Inert Mass(kg)** |
|---|---|---|
| Storable | 7789.4 | 12458.6 |
| Methane | 8064.3 | 13242.5 |
| Storable w/ Tech | 7889.4 | 12458.6 |
| Methane w/ Tech | 6514.3 | 10422.2 |

not just a random occurrence among these four select architectures. When looking at the over 18,000 cases evaluated for this experiment, similar shifts in the architecture types in the optimal objective space may be observed when introducing technologies. Figure 52 shows the distribution of architectures with a given main propulsion system propellant type which exist on the Pareto front of the objective space when no technologies are included. As a result, storable architectures with an NTO/MMH-based main propulsion system account for 73% of the architectures in the Pareto front of the objective space, while cryogenic architectures with a LOX/$LCH_4$-based main propulsion system account for 16%, and cryogenic architectures with a LOX/$LH_2$-based main propulsion system account for only 9%. However, when active cryocooling technology is introduced, the distribution of these architectures on the Pareto front of the objective space shifts, as seen in Figure 53. Here, the cryogenic propellant based architectures now account for a majority of the Pareto front. This shows that the examples seen by the four selected architectures was not just a random occurrence, but a repeated trend: technologies have a marked impact on the resulting objective space and the types of architectures on the Pareto front.

However, this is the case with a single specific technology. What about a different single technology? Figure 54 shows the result of applying advanced composite materials to tanks. Here, there is no shift in the resulting composition of the Pareto front of the objective space. This is because the technology has a similar impact on all architectures in the space. All architectures have propellant tanks and they all

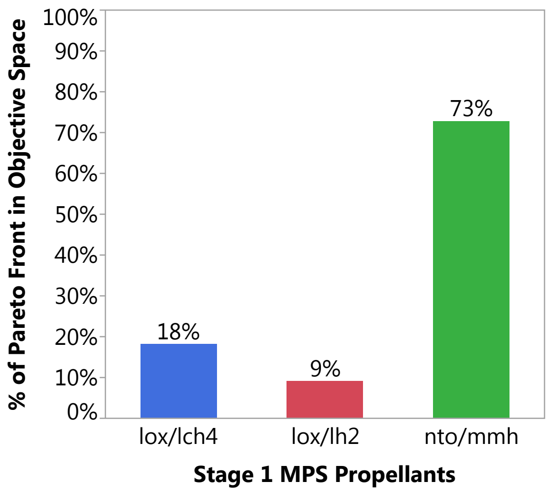


**Figure 52:** Experiment 1 Pareto Optimal Architectures (No Technologies)

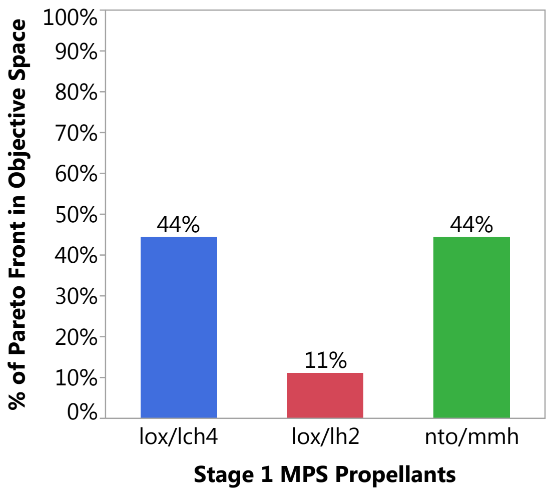


**Figure 53:** Experiment 1 Pareto Optimal Architectures (Active Cryocooling)

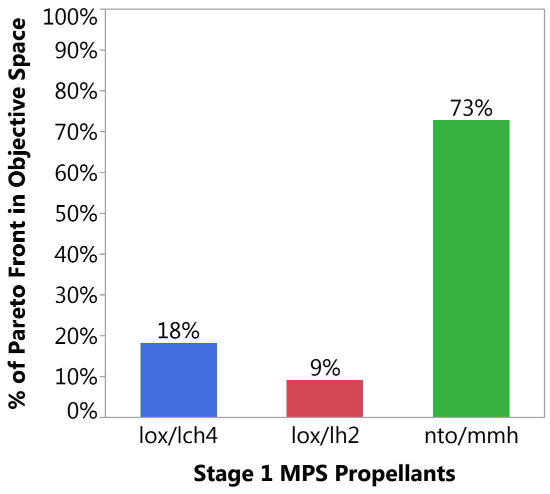


**Figure 54:** Experiment 1 Pareto Optimal Architectures (Composite Tanks)

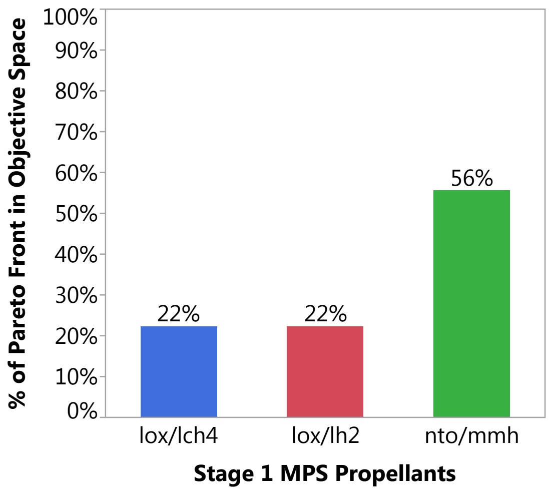


**Figure 55:** Experiment 1 Pareto Optimal Architectures (All Technologies)

gain the same benefits from applying the technology, and as a result, no shift occurs. For this technology, evaluating technologies before down-selection does not make a difference. In fact, it would cost additional computation time. Finally, consider the case of activating all technologies simultaneously, with the results shown by Figure 55. Here, it seems logical to assume that because more technologies are activated, there will be greater shifts in the composition of architectures in the Pareto front of the objective space. However, this is not the result. By activating all technologies, there is indeed a shift in the composition of the Pareto front, but it is muted in comparison to the effect seen in Figure 53 by applying a select, single technology. This is because technologies, such as composite materials for tanks, which have a uniform effect, mask the effects of other technologies, like active cryocooling.

#### *5.1.2.3 Conclusion*

The results observed through the notional example were confirmed to occur with more detailed models which are able to account for interactions among subsystems. This proves that there are at least a few cases where down-selection before technology assessment may indeed exclude a truly optimal architecture from the objective space. Not performing technology evaluation alongside architecture down-selection would have resulted in selection of the suboptimal design. This result was also shown to be a trend across the entire objective space and not just a select few architectures, resulting in dramatic shifts in the composition of the Pareto front of the objective space. However, this is not the case with every technology and set of technologies. It was shown that technologies have an inconsistent, and sometimes unpredictable effect on the resulting objective space. Had technologies not been considered alongside the down-selection of architectures, improper conclusions regarding the most preferred design would be made. Due to these results, Hypothesis 3 shall be accepted.

### 5.1.3 Experiment 2: Testing Individual Results Scheme

**Research Question 5.1**

Would utilizing an individual architecture presentation scheme prevent high-level effects of architecture design decisions from being observed?

As was discussed in Chapter 1.2.3 and shown through a notional example in Chapter 3.3.2, combinatorial explosion will result in very large design spaces with many architectures. The purpose of experiment two is to determine if presenting results of individual architectures for these large spaces would prevent high-level design decisions from being studied. It is expected that high-level architecture trends will be difficult to observe, if at all, due to such large numbers of a single architecture type in the optimal objective space, resulting in poor cross sections of the original architecture space. These ideas are summed up by the following hypothesis:

**Hypothesis 5.1**

The presentation of individual architectures will obscure high-level effects due to flooding of the top results with similar individual designs.

#### *5.1.3.1 Procedure*

The architecture space for experiment two contains 8,921,088 unique architectures consisting of all two stage vehicle variations from the architecture space presented in Table 14, along with all mission and technology options from Table 15 and Table 16 respectively. Data for these alternatives was generated utilizing the digital test bed described in Section 5.1.1. A baseline architecture to determine the similarity value of all other architectures was selected from this architecture space with no active technologies. Utilizing the similarity metric described in Chapter 4.5.4, an experiment is then formulated in which distributions of the similarity of the top performing architectures can be quantified and observed. There are many ways to determine the top performing architectures, either a single objective figure of merit or a multi-objective method. In a multi-objective scenario, the top architectures form a frontier of optimal points, called a Pareto front. These points are all considered optimal as they are undominated by any other point in the objective space. However, the "best" design depends on the relative weightings of the various figures of merit of the multi-objective space. Also, the number of points on the Pareto front is not selectable, as the front is purely based on the results of analysis of the architecture space. In order to vary the number of individual architectures being observed, multiple layers of the Pareto front are observed, known as a Layered Pareto Front (LPF). For details on analyzing an LPF, refer to Appendix E. The following procedures were applied for this experiment:

1. The combined number of alternatives in the problem shall be on the order of 1 million, varying mission, vehicle, and technology options of the architecture space, allowing the potential for individual architecture options to flood the top results.

2. Multiple objective spaces will be analyzed to observe effects due to different sets of figures of merit on the measures of architecture similarity distributions. Single objective spaces consisting of a total mass and cost metric will be evaluated, along with a two-dimensional multi-objective space consisting of both total mass and total cost simultaneously with even weightings.

3. The number of alternatives returned will range from [2:100000] architectures. For a multi-objective space, this requires varying the number of layers in the LPF to achieve the desired number of points.

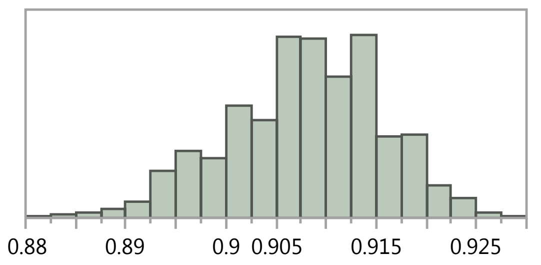
4. Distributions of the similarity of architectures will be recorded and summarized

for each objective type and number of alternatives in the objective space.

Distributions of the similarity metric of the top architectures have narrow summary statistics compared to those of a distribution of the similarity metric of all architectures in the objective space. This implies that a poor cross section of the original objective space is being observed and any effects in the objective space due to high-level architecture decisions will be difficult to observe, supporting Hypothesis 5.1. However, distributions with summary statistics similar to those of a distribution of all architectures in the objective space implies that there is a broad sampling of architecture concepts in the observed objective space that are representative of the entire objective space. This implies that observed effect due to high-level architecture design decisions will be well-represented, refuting Hypothesis 5.1.

#### *5.1.3.2 Results*

The distribution of architecture similarities for all cases is relatively normal with a mean of 0.9079 and a standard deviation of 0.007495. The maximum similarity value in the objective space is 0.9285 and the minimum is 0.8805, resulting in a total range of 0.4795. Through analysis of the results, it was determined that the smallest variation in similarity due to any single attribute is 0.0012324. All technology options account for 0.008627 variation in similarity where each technology option accounts for 0.0012324 variation in the similarity metric. All mission options account for 0.0024648 variation in similarity where each mission option accounts for 0.0012324 variation in similarity. The vehicle space options accounts for the remaining 0.4684 variation of similarity. It is clear that the vehicle space accounts for a large majority of architecture variation, primarily due to combinatorial explosion when including multiple stages. These statistics and observations are the baseline for evaluating the summary statistics while varying the numbers of Pareto fronts and dimensionality of the objective space, and are summarized in Figure 56.



| Quantiles | |
|---|---|
| maximum | 0.928456 |
| minimum | 0.880507 |

| Summary Statistics | |
|---|---|
| Mean | 0.9079202 |
| Std Dev | 0.0074955 |
| Std Err Mean | 2.5095e-6 |
| Upper 95% Mean | 0.9079251 |
| Lower 95% Mean | 0.9079153 |
| N | 8921088 |

**Figure 56:** Objective Space Similarity Distribution

The growth in the total number of observed design points from the objective space with increasing number of layers in the LPF is not consistent across the different objective spaces considered in the experiment, shown in Figure 57. The number of architectures in the LPF depends on the number of layers and the local density of architectures in the objective space for each layer. For the architecture space considered in this experiment, Figure 58 shows the density of architectures along both the total vehicle mass and total vehicle cost metrics. The inconsistency in density of alternatives in the objective space is due to the discrete categorical nature of the architecture and technology spaces. Here, darker regions represent areas in the objective space which are more densely packed with alternatives. These increased densities will result in a greater number of alternatives on a given Pareto front which passes through that region of the objective space. Each Pareto front passes through different regions within the density plot. Because the density of architectures along each dimension of the objective space is not uniform, the relationship between the number of layers and the number of observed architectures is highly nonlinear, as indicated by Figure 57.

Though increasing the number of observed design points may help to give a more representative set of architectures from the objective space, it alone does not definitively imply a more representative cross section of the entire objective space. The designs may still be highly focused around a specific subset of architectures. It is important to consider other metrics when determining the quality of the observed

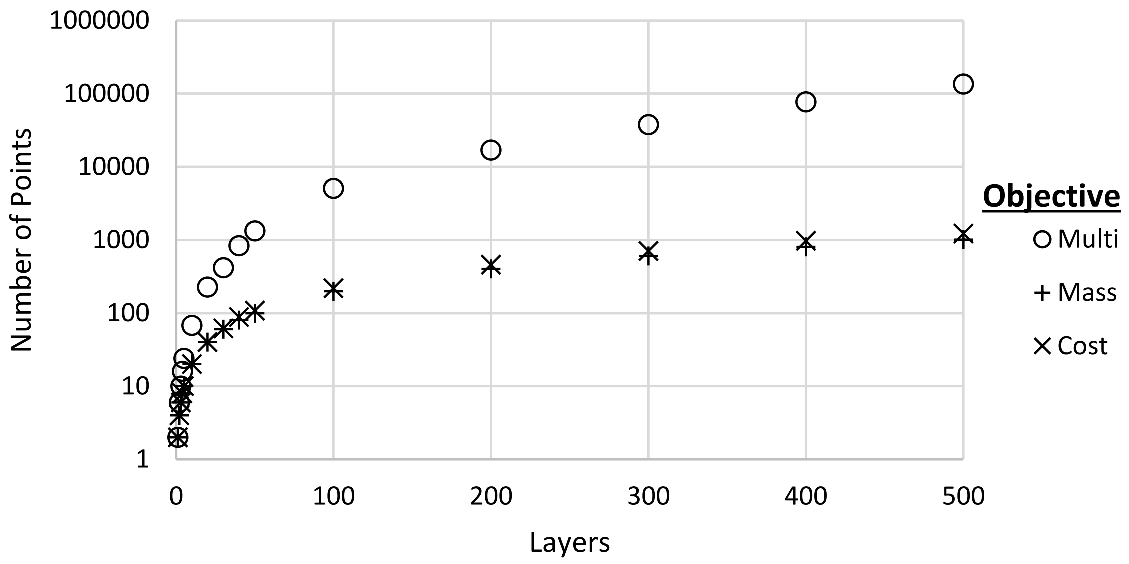


**Figure 57:** Relationship of Number of Pareto Front Layers to Total Number of Data Points in the Similarity Distribution

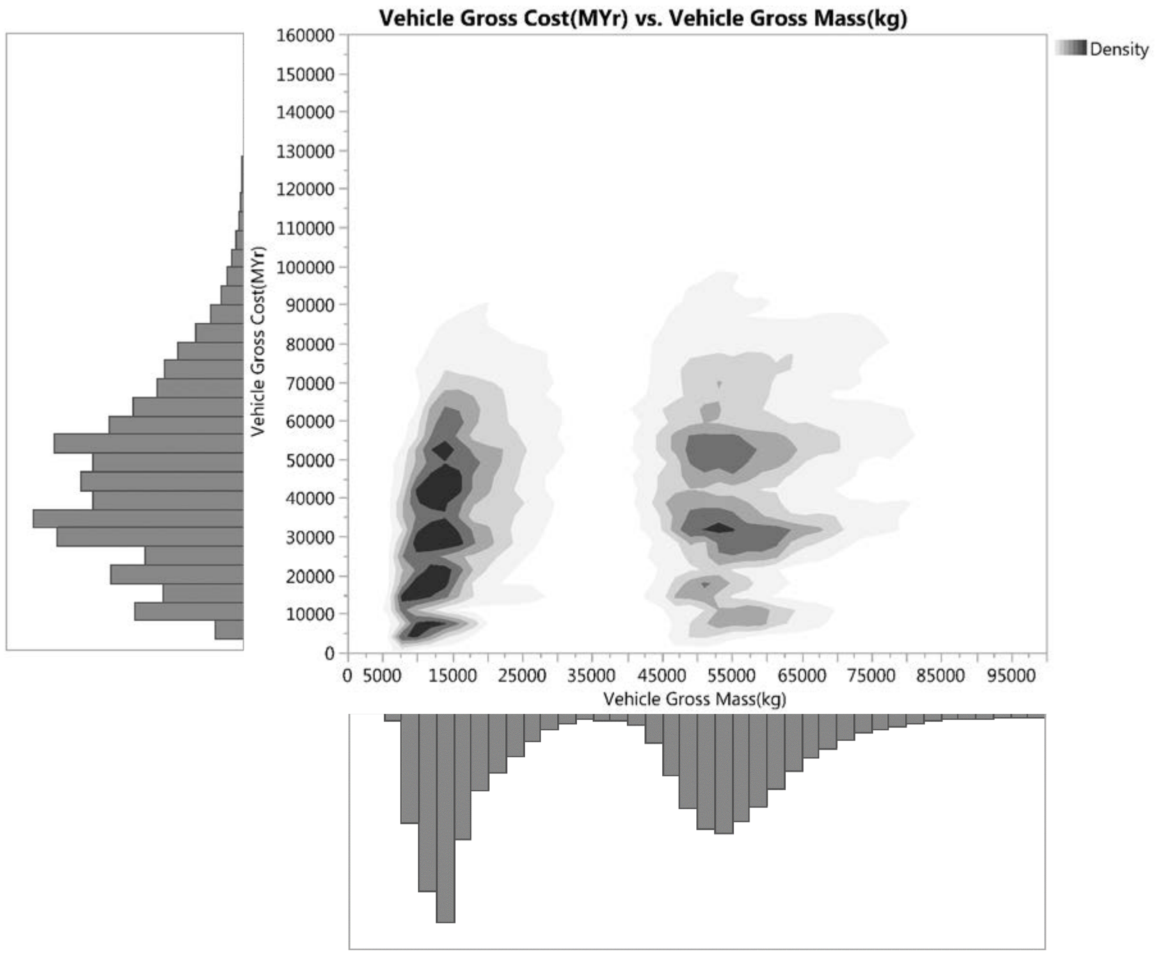


**Figure 58:** Objective Space Architecture Density

portion of the objective space being represented by the subset of architectures on an LPF. Collectively, the subset distribution's mean, standard deviation, minimum, maximum, and range of architecture similarity can provide indications as to the quality of representation of all cases in the objective space. However, because the number of architectures contained in the subset is not consistent with the number of layers in the LPF across different objectives, similarity distribution metrics are plotted against the total number of architectures in the LPF.

All distribution parameters of the set of architectures in the LPF appear to approach the parameters of the baseline objective space distribution of architecture similarity in a logarithmic trend. For each of the distribution metrics, the specific objective has little effect on the variation in the metric, as is visible by a noticeable overlap in the data for the three different objectives considered in this experiment, shown in Figure 59 through Figure 63. The solid red line in these plots is the metric value of the baseline distribution for all architectures, while the dashed red line represents the fifty percent mark between the metric for a distribution of an LPF with two architectures and the baseline similarity distribution's metric. Obviously, this low number of architectures in the observed objective space would be a very poor representation of the whole objective space and provides a lower boundary for determining these 50% marks.

The minimum value of the distribution of architecture similarity with respect to the number of observed architectures in the LPF is represented in Figure 59. The baseline similarity distribution has a minimum of 0.880507 while an LPF of one layer has a minimum of 0.893173. The 50% value between the minimum similarity of these two distributions is 0.88684. The data shows that an LPF containing on the order of 100 points is able to reduce a majority of the difference in minimum similarity between a distribution of an LPF containing two points and the baseline distribution.

The maximum value of the distribution of architecture similarity with respect to

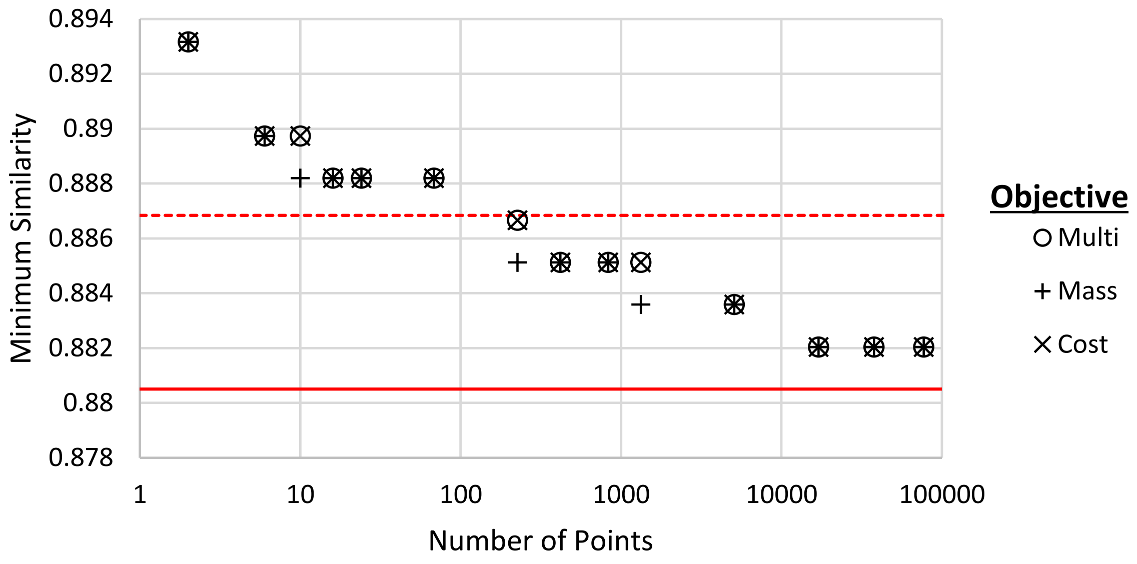


**Figure 59:** Relationship of Number of Pareto Front Layers to Similarity Distribution Minimum

the number of observed architectures in the LPF is represented in Figure 60. The baseline similarity distribution has a maximum of 0.928456 while an LPF with one layer has a maximum of 0.894712. The 50% value between the minimum similarity of these two distributions is 0.911584. An LPF containing on the order of 1,000 points is able to reduce a majority of the difference in maximum similarity between a distribution of an LPF containing two points and the baseline's distribution.

The range of the distribution of architecture similarity with respect to the number of observed architectures in the LPF is represented in Figure 61. The baseline similarity distribution has a range of 0.047949 while an LPF with one layer has a range of 0.001539. The 50% value between the range of similarity of these two distributions is 0.024744. An LPF containing on the order of 50 points is able to reduce a majority of the difference in the range of similarity between a distribution of an LPF containing two points and the baseline distribution.

The mean of the distribution of architecture similarity with respect to the number of observed architectures in the LPF is represented in Figure 62. The baseline similarity distribution has a mean of 0.90792 while an LPF with one layer has a mean of

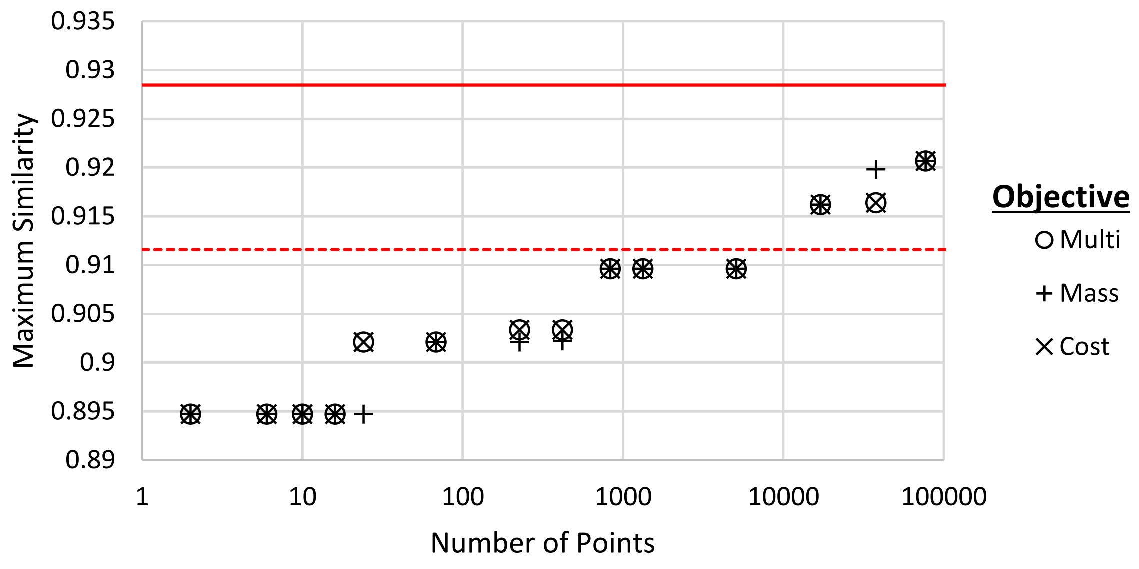


**Figure 60:** Relationship of Number of Pareto Front Layers to Similarity Distribution Maximum

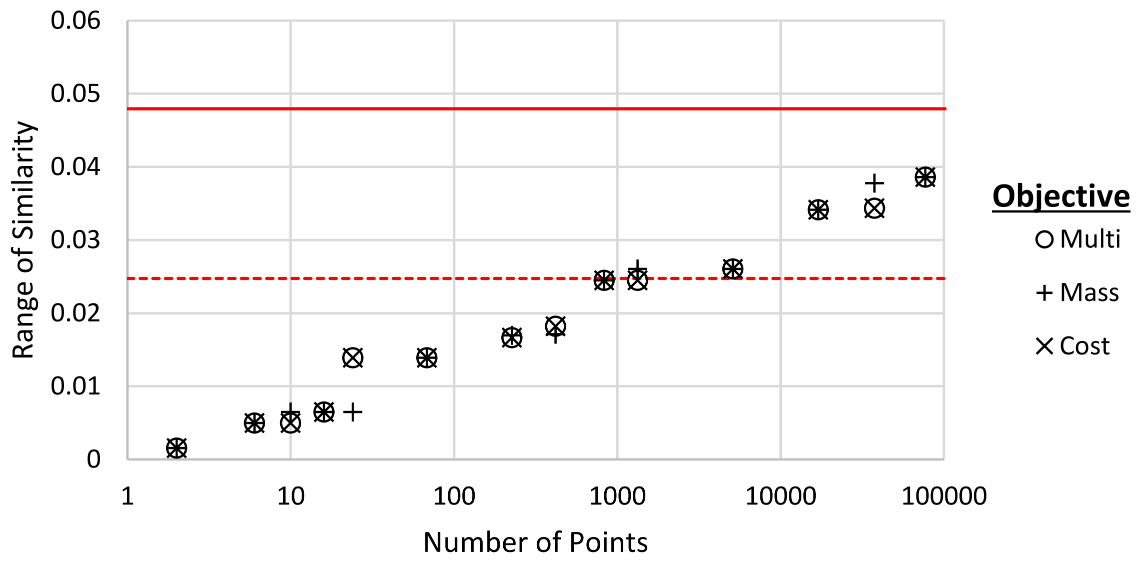


**Figure 61:** Relationship of Number of Pareto Front Layers to Similarity Distribution Range

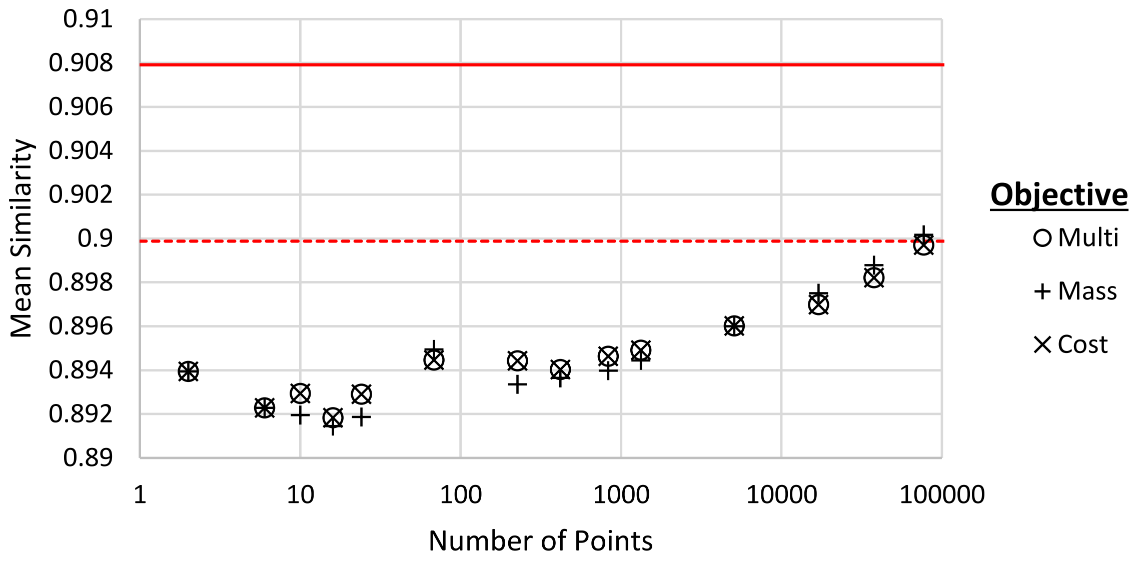


**Figure 62:** Relationship of Number of Pareto Front Layers to Similarity Distribution Mean

0.893943. The 50% value between the range of similarity of these two distributions is 0.899881. An LPF containing on the order of 100,000 points is able to reduce a majority of the difference in the mean of similarity between a distribution of an LPF containing two points and the baseline distribution. The shift in mean towards the baseline is slower with increased number of points compared to the other metrics considered.

The standard deviation of the distribution of architecture similarity with respect to the number of observed architectures in the LPF is represented in Figure 63. The baseline similarity distribution has a standard deviation of 0.007496 while an LPF with one layer has a standard deviation of 0.001088. The 50% value between the standard deviation of similarity of these two distributions is 0.004292. An LPF containing on the order of 100 points is able to reduce a majority of the difference in the standard deviation of similarity between a distribution of an LPF containing two points and the baseline distribution.

The full set of data analyzed in this experiment can be found in Appendix F, including distributions for all of the LPFs tested in this experiment.

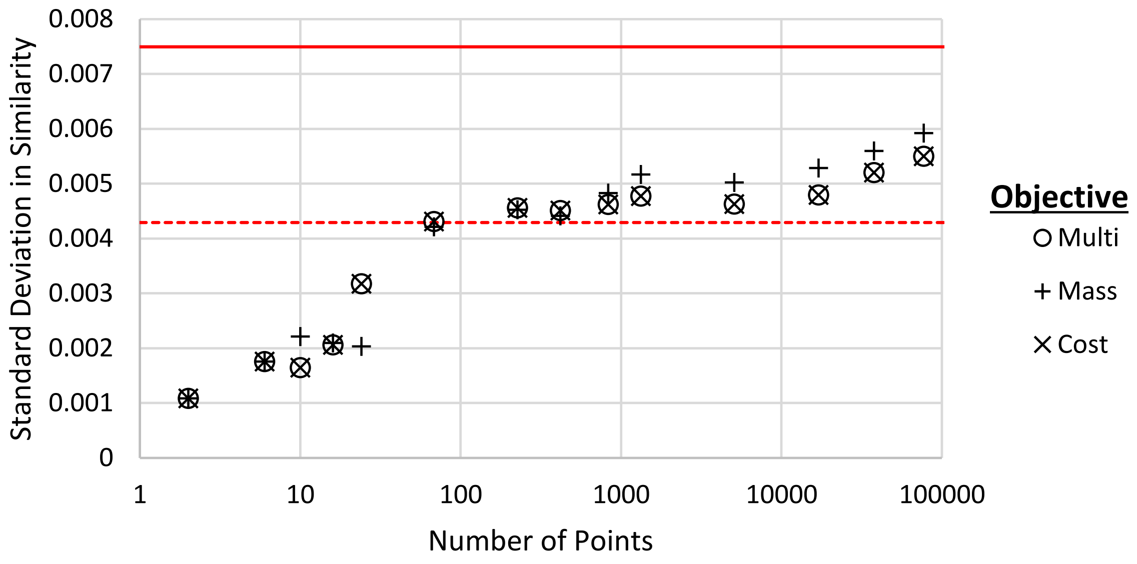


**Figure 63:** Relationship of Number of Pareto Front Layers to Similarity Distribution Standard Deviation

*5.1.3.3 Conclusion*

The results from this experiment indicate that flooding of top results of the objective space can occur, represented by large deviations in the summary statistics of the distributions of architecture similarity between the subset of top results and the entire objective space. Observation of extremely small subsets of the results in the objective space will result in poor representation of the true objective space. These observations result in the acceptance of Hypothesis 5.1. However, this observation is limited to small subsets of the objective space. It is possible to increase the quality of representation of the original objective space by increasing the total number of architectures in the top subset. Overall, it was determined that the representation of the entire objective space by the subset of the top N architectures increases in a logarithmic trend. Relatively low numbers of top architectures from the objective space may adequately represent the entire objective space. These observations were found to be true across various objectives with dissimilar distributions of points within individual figures of merit. Further work is needed to confirm these results across higher dimensional objective spaces spanning additional problem domains.

### 5.1.4 Experiment 3: Portfolio Grouping Criteria

**Research Question 5.2**

How do the grouping criteria used for forming portfolios of architectures affect the variance of the resulting portfolios?

The purpose of experiment 3 is to determine how grouping parameters on which to form sets of architectures affect the resulting figures of merit of those portfolios. This will help guide a more structured approach to selecting grouping criteria for large architecture spaces to aid in creating clear and concise presentation of results, while minimizing loss of information about the overarching objective space.

**Hypothesis 5.2**

Variance of the objective metrics within and between portfolios will correlate positively with the size of the portfolios, measured by the number of grouped architectures.

#### *5.1.4.1 Procedure*

The architecture space for experiment three consists of all vehicle variations from the architecture space presented in Table 14, along with all mission and technology options from Table 15 and Table 16 respectively. Again, data for each of the architectures was generated utilizing the digital test bed described in Section 5.1.1. The objective space is filtered to architectures with a total vehicle gross mass of no more than 100,000 kg to prevent outlier architectures from skewing observed distributions. To construct portfolios of varying size, discrete architecture space and technology space options will be selected independently as grouping criteria to reduce interactions between the two spaces. Within the architecture space, options relating to the vehicle and mission space will be selected independently to reduce interaction effects between the two groups of architecture options. The selection of these grouping criteria shall be selected at random from their respective spaces.

Variance between portfolios will be measured through numerous objective space metrics: architecture similarity, vehicle total mass in kilograms (kg), vehicle propellant mass fraction (PMF), and vehicle total cost in man-years (MYr). The variance in the objective space metrics of the architectures within a portfolio, as well as the variance in the aggregate objective space metrics between each portfolio, will be analyzed. Large variances between portfolio-level aggregate objectives, along with small variances between architectures within a portfolio for small portfolios support the claims of Hypothesis 5.3.

##### *5.1.4.2 Results*

Figure 64 show the variance in architecture similarity, vehicle gross mass, vehicle gross cost, and vehicle PMF for portfolios formed from vehicle options in the architecture space. Portfolios of varying size are formed from various sets of the following randomly selected vehicle options:

- Tank Configuration
- Structures Type
- Power System
- MLI Layers

Figure 64 shows a positive correlation in variance of the architecture similarity across the vehicle portfolios, denoted by the slope of the 95% ellipse around the data. Variance in the architecture similarity between portfolios appears to be positively correlated with the size of the portfolio. This is observed by the relative distance between points for a given portfolio size. Both of these observations support Hypothesis 5.3. However, these observation do not hold for other metrics in the objective space. Variance in the total vehicle gross mass, total vehicle cost, and vehicle PMF all seem to have no correlation with the size of portfolios, denoted by the nearly zero slope of the 95% ellipse around these sets of data. Moreover, the observation of variance between portfolios does not hold true, as there are large differences in variance between portfolios of a given size, large and small. This is best observed in the variance of vehicle gross cost, where there is a relatively large difference in the variance regardless of portfolio size. These observations are in clear opposition to Hypothesis 5.3. The distribution summary statistics for each of the portfolios is provided in Table 60 of Appendix G.

Figure 100 in Appendix G shows the variance in architecture similarity, vehicle gross mass, vehicle gross cost, and vehicle PMF for portfolios formed from mission options in the architecture space. Portfolios of varying size are formed from various sets of the following mission options:

- Destination Duration
- Inbound Correction Maneuver

Similarly, Figure 101 in Appendix G shows the variance in architecture similarity, vehicle gross mass, vehicle gross cost, and vehicle PMF for portfolios formed from technology packages in the technology space. Portfolios of varying size are formed via the following criteria:

- No Active Technologies
- Single Active Technology
- All Technologies Active

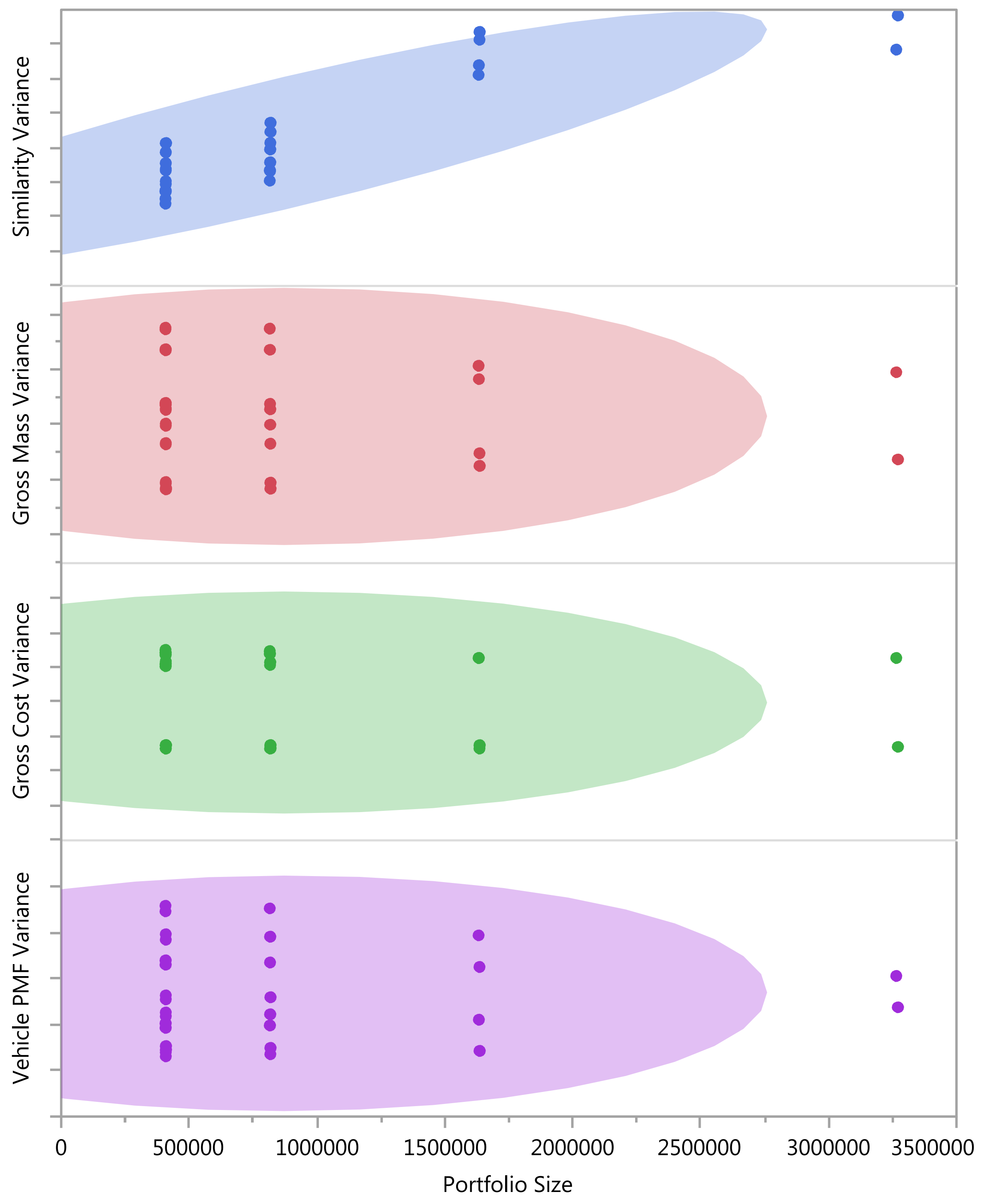


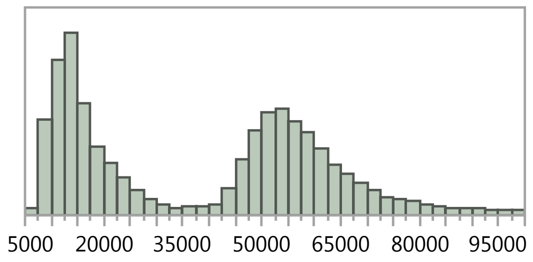
**Figure 64:** Correlation in Portfolio Size vs Objective Metric Variance for Vehicle-based Portfolios

Data for these portfolio schemes show similar observations, refuting Hypothesis 5.3. Many of the metrics show very little to no correlation between portfolio size and variance of a given metric. Some show even a slight negative correlation, in direct opposition to Hypothesis 5.3. Also, differences in the variance of a given metric do not seem to be correlated to the size of the portfolio. The distribution summary statistics for the mission and technology portfolios are provided in Table 58 and Table 59 of Appendix G respectively.

It is apparent that the variation in a given objective space metric for the architectures contained in a portfolio is not necessarily correlated to the size of a portfolio. Sizes of portfolios are a result of the various grouping criteria, but have little to do with the variation of objective metrics. Rather, the variation is due to the grouping criteria which form the portfolios themselves. Figure 66 and Figure 67 illustrate the distribution of architecture vehicle gross mass for portfolios formed by varying the payload mass in the architecture space. These two distributions clearly show that vehicle payload mass is the primary cause of the bimodal distribution of vehicle gross mass across all architectures observed in Figure 65.

Similar observations can be made in the distributions of architecture similarity across different portfolio grouping criteria. Figure 68 is the distribution of architecture similarity for the entire architecture space of this experiment, while Figure 69 is the distributions for all single stage vehicles and Figure 70 is the distributions for all two stage vehicles. Again, it is clear that the number of stages in the vehicle has a large impact in skewing the distributions of architecture similarity in the objective space. However, unlike in the portfolios formed by payload mass options, these portfolios exhibit drastically different portfolios sizes. This is because there are many more combinations of two stage vehicles than single stage vehicles in the architecture space. In fact, the single stage portfolio has a lower variance in architecture similarity than the larger two stage vehicle portfolio, in direct opposition to Hypothesis 5.3.

**Vehicle Gross Mass(kg)**



**Quantiles**

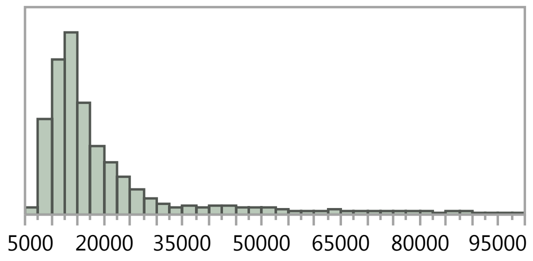
| | |
|---|---|
| maximum | 100000 |
| minimum | 6290.82 |

**Summary Statistics**

| | |
|---|---|
| Mean | 38151.693 |
| Std Dev | 23058.586 |
| Std Err Mean | 9.0189858 |
| Upper 95% Mean | 38169.369 |
| Lower 95% Mean | 38134.016 |
| N | 6536570 |

**Figure 65:** Total Vehicle Mass(kg) Distribution of All Architectures

**Vehicle Gross Mass(kg)**



**Quantiles**

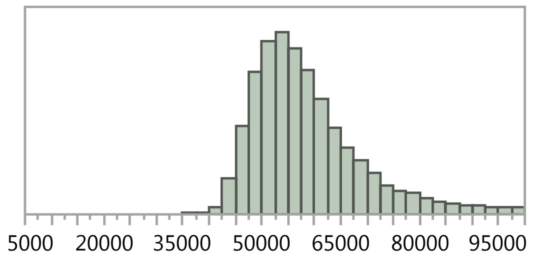
| | |
|---|---|
| maximum | 100000 |
| minimum | 6290.82 |

**Summary Statistics**

| | |
|---|---|
| Mean | 18887.709 |
| Std Dev | 13195.302 |
| Std Err Mean | 7.2041085 |
| Upper 95% Mean | 18901.829 |
| Lower 95% Mean | 18873.589 |
| N | 3354889 |

**Figure 66:** Total Vehicle Mass(kg) Distribution of Architectures with 1,000 kg Payload

**Vehicle Gross Mass(kg)**



**Quantiles**

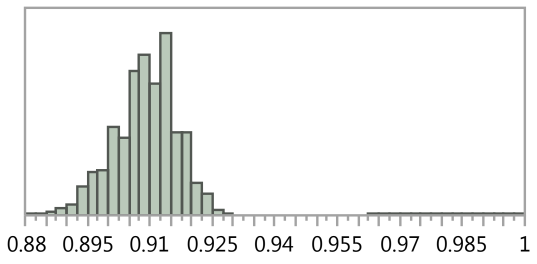
| | |
|---|---|
| maximum | 100000 |
| minimum | 35932.2 |

**Summary Statistics**

| | |
|---|---|
| Mean | 58464.391 |
| Std Dev | 10239.04 |
| Std Err Mean | 5.7402516 |
| Upper 95% Mean | 58475.642 |
| Lower 95% Mean | 58453.14 |
| N | 3181681 |

**Figure 67:** Total Vehicle Mass(kg) Distribution of Architectures with 10,000 kg Payload

**Similarity**

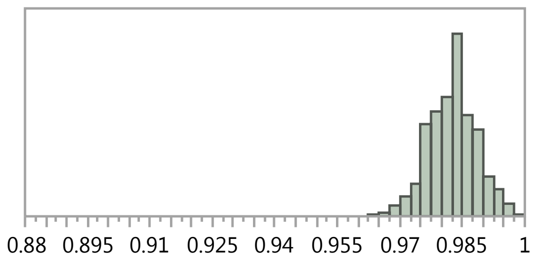


| Quantiles | |
|---|---|
| maximum | 1 |
| minimum | 0.880507 |

| Summary Statistics | |
|---|---|
| Mean | 0.9091657 |
| Std Dev | 0.0082428 |
| Std Err Mean | 3.2241e-6 |
| Upper 95% Mean | 0.909172 |
| Lower 95% Mean | 0.9091594 |
| N | 6536570 |

**Figure 68:** Architecture Similarity Distribution of All Architectures

**Similarity**

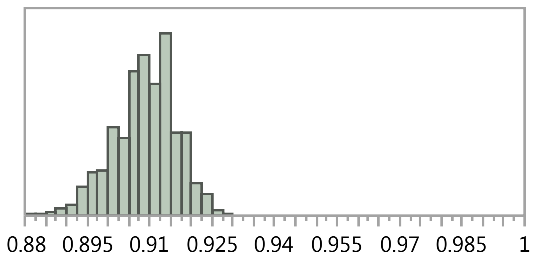


| Quantiles | |
|---|---|
| maximum | 1 |
| minimum | 0.963096 |

| Summary Statistics | |
|---|---|
| Mean | 0.9825808 |
| Std Dev | 0.0055789 |
| Std Err Mean | 4.0358e-5 |
| Upper 95% Mean | 0.9826599 |
| Lower 95% Mean | 0.9825017 |
| N | 19109 |

**Figure 69:** Architecture Similarity Distribution of Architectures with 1 Stage

**Similarity**

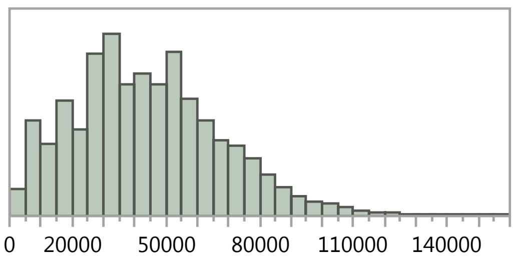


| Quantiles | |
|---|---|
| maximum | 0.928456 |
| minimum | 0.880507 |

| Summary Statistics | |
|---|---|
| Mean | 0.9089505 |
| Std Dev | 0.0072252 |
| Std Err Mean | 2.8302e-6 |
| Upper 95% Mean | 0.908956 |
| Lower 95% Mean | 0.9089449 |
| N | 6517461 |

**Figure 70:** Architecture Similarity Distribution of Architectures with 2 Stages

**Vehicle Gross Cost(MYr)**

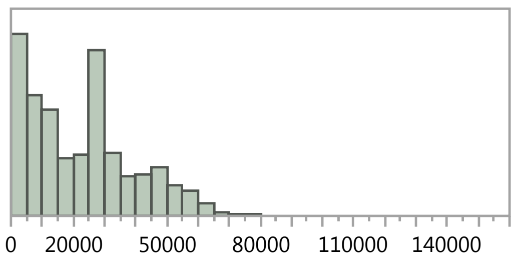


| Quantiles | |
|---|---|
| maximum | 158866 |
| minimum | 730.255 |

| Summary Statistics | |
|---|---|
| Mean | 43722.196 |
| Std Dev | 22892.64 |
| Std Err Mean | 8.9540788 |
| Upper 95% Mean | 43739.746 |
| Lower 95% Mean | 43704.647 |
| N | 6536570 |

**Figure 71:** Total Vehicle Cost(kg) Distribution of All Architectures

**Vehicle Gross Cost(MYr)**

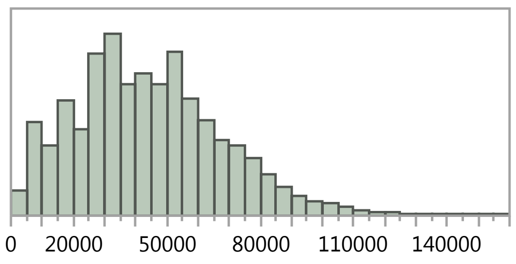


| Quantiles | |
|---|---|
| maximum | 3423.25 |
| minimum | 1460.78 |

| Summary Statistics | |
|---|---|
| Mean | 21837.52 |
| Std Dev | 16443.695 |
| Std Err Mean | 118.95438 |
| Upper 95% Mean | 22070.681 |
| Lower 95% Mean | 21604.359 |
| N | 19109 |

**Figure 72:** Total Vehicle Cost(kg) Distribution of Architectures with 1 Stage

**Vehicle Gross Cost(MYr)**



| Quantiles | |
|---|---|
| maximum | 158866 |
| minimum | 1460.78 |

| Summary Statistics | |
|---|---|
| Mean | 43786.362 |
| Std Dev | 22878.121 |
| Std Err Mean | 8.9615087 |
| Upper 95% Mean | 43803.926 |
| Lower 95% Mean | 43768.797 |
| N | 6517461 |

**Figure 73:** Total Vehicle Cost(kg) Distribution of Architectures with 2 Stages

Trends in the full objective space may be difficult to observe due to the large number of alternatives in the objective space from a subset of the architecture space. However, by grouping architectures into portfolios, observations which could not be made in the full objective space with all architectures combined may now become visible. This type of behavior can be seen by again forming portfolios based on the number of vehicle stages and observing the distributions of vehicle gross cost and vehicle PMF. Figure 71 shows the distribution of vehicle gross cost in the objective space for all architectures. Because the two stage vehicle options are so numerous compared to the single stage vehicle options, the two stage vehicle options drive this distribution, as is seen by comparing the distributions in Figure 73 and Figure 71. However, by observing the distributions of vehicle gross cost for only single stage vehicles, shown by Figure 72, one can see a clear skew towards lower cost for single stage vehicles compared to two stage vehicles. Similar observations may be made in vehicle PMF with vehicle number of stages. A single stage vehicle will typically have a higher PMF compared to a typical two stage vehicle. Logically, this makes sense, considering a two stage vehicle will require two sets of engines, plumbing, tanks, structures, etc. which drive the PMF of the whole vehicle down. Distributions illustrating this observation are shown by Figure 102 through Figure 104 in Appendix G.

Portfolios of the technology space is a common way of viewing the objective space. Table 21 shows the mean boiloff rate of the first and second stages in kilograms of propellant per day for each of the technology portfolios listed. Observation of these portfolio grouping criteria provides insight into which technology, or group of technologies may aid in reducing boiloff of cryogenic propellant for space vehicles. Here, the active cryocooling technology reduces the boiloff rate of both stages to zero, while the portfolio containing all technologies will obviously have the same benefit of the single technology, provided no negative interactions exist.

**Table 21:** Technology Impact on Stage Boiloff Rate

| Portfolio Description | Mean Stage 1 Boiloff Rate(kg/day) | Mean Stage 2 Boiloff Rate(kg/day) |
|---|---|---|
| No Technologies | 10.024 | 5.317 |
| Wireless Sensors | 10.017 | 5.317 |
| Composite Structures | 9.600 | 5.935 |
| Composite Tanks | 9.969 | 5.322 |
| Active Cryocooling | 0.0 | 0.0 |
| Integrated MPS/RCS | 10.052 | 5.312 |
| Low Leak Valves | 9.912 | 5.277 |
| High Capacity Batteries | 9.960 | 5.311 |
| All Technologies | 0.0 | 0.0 |

##### *5.1.4.3 Conclusion*

The observation of the distributions of objective space metrics for portfolios of architectures across different grouping criteria showed that the size of the resulting portfolios has little impact on the resulting variance of those objective space metrics. Any observed correlation is coincidental in nature. Variation in these metrics between portfolios also was observed to be unrelated to the size of the portfolios. However, further observation of the data from this experiment provided valuable insight into trends in the objective space with regard to the physical architecture space. By forming portfolios of architectures from these physical architecture space options, new trends in metrics such as gross mass, gross cost, similarity, and vehicle PMF were observable. However, the grouping criteria utilized for observing results may lead to different conclusions regarding the effects of technologies on architectures. Only focusing on a single portfolio grouping criterion will result in conclusions being made without full understanding of the trends which may exist in the problem. The results from this experiment lead to the rejection of Hypothesis 5.3, but also leads to the following conjecture:

**Conjecture 5.2**

Variance in the objective space metrics within and between portfolios are influenced by the grouping criteria derived from options in the architecture space.

### 5.1.5 Experiment 4: Testing Portfolio Results Scheme

**Research Question 5.3**

Would utilizing a portfolio scheme for grouping architectures obscure high-performing outlier architectures?

The purpose of Experiment 4 is to determine if implementing a portfolio scheme for grouping architectures when large numbers of alternatives exist has the potential of obscuring optimal designs. This could occur if the highest performing design is hidden within a portfolio with an aggregate performance less than another portfolio, as was shown by a notional example during the formulation of the research question and hypothesis. However, testing all potential grouping criteria to be certain that no obscuring will exist is impractical, and as such, a null hypothesis, shown below, was formed where only one case of design obscuring is enough to disprove the null hypothesis, leading to the conclusion that obscuring of design may indeed exist.

**Hypothesis 5.3**

High-performing outlier architectures will not be obscured using a portfolio evaluation scheme because they will be contained in a portfolio with other similar architectures which will exhibit similar behavior, raising the performance of the entire portfolio.

#### *5.1.5.1 Procedure*

The hypothesis assumes that due to the large number of design alternatives in such a large and complex space, there will not be single outlier designs. Rather, there will be groups of designs. It would be logical to assume that these collections of outlier designs would be grouped together in a portfolio scheme due to their relative similarity in physical systems and/or operations. These natural groups of outliers would bring the performance of the entire portfolio up.

This experiment utilizes the same architecture space from Experiment 3, consisting of all vehicle variations from the architecture space presented in Table 14, along with all mission and technology options from Table 15 and Table 16 respectively. This results in the same data set utilized in Experiment 3. Again, the objective space is filtered to architectures with a total vehicle gross mass of no more than 100,000 kg to prevent under-performing and oversized architecture outliers from skewing observed distributions. The metrics for which portfolios and the optimal design will be evaluated against are total vehicle mass in kilograms (kg), total vehicle cost in man-years (MYr), and vehicle PMF. Initially, the results of Experiment 3 will be examined for any cases resulting in a single optimal architecture being obscured in a lower performing portfolio. If this occurs, then Hypothesis 5.3 will be rejected, thus proving that optimal designs may become obscured when grouping architectures into portfolios.

However, if the results of Experiment 3 prove inconclusive with regard to this research question, further experimentation will be performed in an attempt to observe an obscured optimal design. Variations to the portfolio grouping scheme will be selected by combining related selections from multiple spaces simultaneously. If these further cases still do not result in optimal design obscuring, sets of randomly selected grouping criteria based on architecture and technology space options combined will be selected in an attempt to observe an obscured design.

#### *5.1.5.2 Results*

Based on the architecture space for this experiment, the single-objective optimal designs for each of the three figures of merit are listed below:

**Mass:** two stage all solid propellant architecture with all technologies applied resulting in a total vehicle mass of 6290.82 kg

**Cost:** single stage solid propellant architecture with all technologies applied resulting in a total vehicle cost of 730.3 MYr

**PMF:** single stage $LO_2/LCH_4$ propellant architecture with the composite structures technology applied resulting in a PMF of 0.866

Results from experiment 3 provided one such case of obscured optimal designs within suboptimal portfolios. Nine portfolios result when the objective space is broken into portfolios based on the following grouping criteria:

- No Active Technologies
- Single Active Technology
- All Technologies Active

The nine portfolios along with their means for each of the figures of merit in the objective space are provided in Table 22. Here, the best portfolio for reducing mass is to apply all available technologies, the best portfolio for reduced cost is to apply no technologies, and the best portfolio to increase PMF is to apply the composite structures technology.

Observation of these results shows that the optimal portfolio to reduce cost is to apply no technologies; however, the best in class architecture for cost is one which applies all technologies. This is likely due to the fact that cost is such a strong function of mass in the TransCost model selected and developed for this body of work.

Table 22: Technology Portfolios Objective Means

| Description | N | $\mu\, m_{gross}$ (kg) | $\mu\, C_{gross}$ (MYr) | $\mu\, PMF$ |
|---|---|---|---|---|
| No Techs | 708506 | 3.894e+4 | 4.009e+4 | 6.612e-1 |
| Wireless Sensors Only | 708554 | 3.890e+4 | 4.417e+4 | 6.613e-1 |
| Composite Structures Only | 719846 | 3.715e+4 | 4.325e+4 | 6.661e-1 |
| Composite Tanks Only | 709442 | 3.857e+4 | 4.402e+4 | 6.617e-1 |
| Active Cryocooling Only | 768338 | 3.861e+4 | 4.634e+4 | 6.309e-1 |
| Integrated MPS/RCS Only | 708338 | 3.924e+4 | 4.104e+4 | 6.612e-1 |
| Low Leak Valves Only | 709001 | 3.843e+4 | 4.336e+4 | 6.571e-1 |
| High Capacity Batteries Only | 708817 | 3.859e+4 | 4.188e+4 | 6.616e-1 |
| All Techs | 795728 | 3.527e+4 | 4.854e+4 | 6.299e-1 |

The cost gain associated with applying technologies for the specific architecture was outweighed by the cost savings resulting from the reduction in overall mass due to those technologies for the specific architecture.

#### *5.1.5.3 Conclusion*

The results from experiment 3 provided at least one case where the optimal design for a specific single-objective metric was obscured within a suboptimal portfolio for that same metric. This results in the rejection of the null hypothesis stated by Hypothesis 5.3, supporting the claim that grouping architectures into portfolios of designs indeed has the potential to obscure designs. Caution must be taken to ensure that high-performing designs of interest are not ignored due to an employed portfolio scheme. Only relying on results presented in portfolios of alternatives has the potential to obscure optimal designs. Rather, exploration of the objective space should be performed across a variety of portfolio schemes, as well as individual alternatives to ensure a full understanding of the objective space and the optimal designs. It is important to note that this experiment focused primarily on one-dimensional objective spaces of various figures of merit. However, with multi-dimensional objective spaces, multiple design points are considered optimal. This complicates the otherwise trivial concept of optimal design obscuring considered in this experiment. Further work is needed to study how designs may be obscured in multi-objective scenarios.

### 5.1.6 Observation: Differences in Figures of Merit

Experiments 1 through 4 can be observed to make note of differences in the metrics being utilized. Over the course of these four experiments defined in this chapter, analysis results have focused on both individual architecture presentation of the data, as well as portfolios of architectures. An observation of the metrics used in these experiments can lead to a conclusion about whether similar metrics are utilized between these two presentation schemes. It is expected that utilizing a form of results grouping will naturally lead to the use of additional metrics relating to group composition and statistical distribution summary metrics.

#### *5.1.6.1 Results*

The experiments presented in this body of work relied heavily on making observations of groups of architectures. Experiment 1 observed the composition of the total objective space by the main propulsion system employed by the architectures to observe shifts in the overarching objective space due to technologies. In Experiment 2, subsets of the objective space were observed by only observing the top designs. This was done on a single-objective and multi-objective basis. In both ways, summary statistics of the distribution of figures of merit for the subsets were used to determine how well a given subset represented the total objective space. Statistics such as minimum, maximum, range, mean, and standard deviation of the distribution of the similarity of architectures of the subsets from the objective space provided insight into the quality of the cross section of the total objective space being observed. Finally, Experiment 3 and Experiment 4 focused on breaking the objective space into portfolios of designs grouped together by a set of grouping criteria. The resulting portfolios were studied by comparing means and variance of specific figures of merit for the architectures contained within a portfolio.

Through these experiments, it is clear that when groups of architectures were being studied or observed, regardless of whether the groups were portfolios or subsets based on varying criteria, each group was summarized through distributions of the constituent architectures. This supports Hypothesis 6, that additional metrics summarizing the performance and composition of groups of architectures need to be included when establishing value of a problem.

Similarly, it is important to note the relative weightings of figures of merit when considering a multi-objective space should be included as additional metrics to consider. The experiments in this research implicitly assumed even weightings to figures of merit, regardless of how groups of architectures may have been formed. However, it is well known that some weighting must be applied to any multi-objective problem, whether it is implicitly assumed or stated during problem formulation.

#### *5.1.6.2 Conclusion*

These observations support the claim made by Conjecture 6, that additional metrics should be considered when studying results as groups of architectures. These metrics should consist of summaries of the figures of merit of the architectures contained in the groups themselves, as well as metrics which describe the composition of a given group and how specific figures of merit were weighted when forming those groups.

## *5.2 Summary of Developed Framework*

From this body of research, a flexible framework is presented, focusing on integrating architecture analysis and technology evaluation at a subsystem level for the purpose of exploring large design spaces in an attempt to better understand the underlying system spaces and their potential interactions. The framework shall be referred to as the Integrated Architecture and Technology Exploration (IntegrATE) framework. The process of the framework is shown in Figure 74. Many of the underlying components of this framework employ previously developed techniques and methods which suit the overarching goal of integrating architecture analysis and technology evaluation at a subsystem level. Concepts and techniques such as quality function deployment, DoD viewpoints, and set theory-influenced system decomposition are utilized to perform many of the steps outlined in the IntegrATE framework. However, IntegrATE does not dictate which methods or techniques shall be utilized for the purpose of flexibility. This section summarizes the basic steps required by the IntegrATE framework developed as a result of the experiments discussed in Section 5.1. The steps that follow will typically be arbitrated by architects who gather the necessary information from technologists and subject matter experts. This will help to eliminate bias in the technologies and architectures represented within a problem, as this architect should represent a neutral party with no particular bias for specific architectures or technologies, but who is primarily interested in providing a broad representation for the problem being studied.

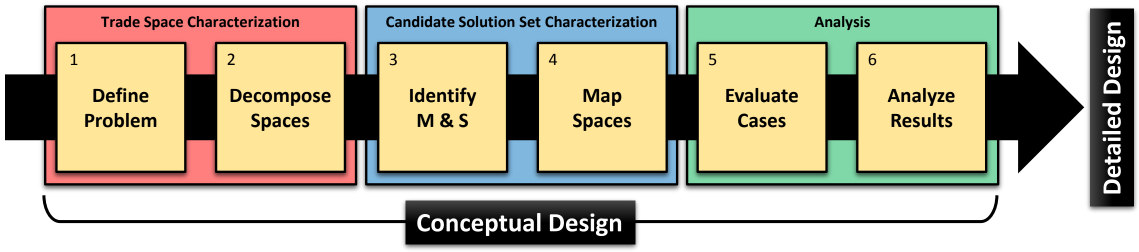


**Figure 74:** IntegrATE Framework Flow Diagram

### 5.2.1 Step 1: Define the Problem

The first step of IntegrATE is to define the problem by taking the requirements from the customer and translating them into quantifiable engineering metrics. Typically, some customer or societal need is what prompts the design of a new product, often referred to as the "voice of the customer" [57]. The customer requirements should capture all aspects of the design including, but not limited to physical performance, budgetary constraints, and schedule. However, typically these requirements are subjective in nature and must be translated into quantifiable metrics, often referred to as the "voice of the engineer". There exist many techniques in literature for performing this task. One such technique employed extensively in many methods is quality function deployment. This technique was chosen for the initial formulation of IntegrATE, but designers could use other methods for translating customer requirements into engineering metrics if preferred.

The objective space of the problem is defined by the engineering metrics mapped to the customer requirements during quality function deployment. Typically this is an n-dimensional space where n is the number of customer requirements tied to engineering metrics. It is important during this step of the framework to capture not only the customer requirements, but also the relative importance of each requirement to the customer. This translates into weighting metrics which will be required when performing multi-objective analysis of the results in step 6. Additionally the translation of customer requirements to engineering metrics should not include any bias toward a specific architecture or technology.

### 5.2.2 Step 2: Decompose Architecture and Technology Spaces

The second step of IntegrATE is to decompose the problem into an architecture space and associated technology space. These spaces define the physical and functional breakdown of potential designs to meet the customer requirements from step 1. Again, the selection of potential architectures and technologies should be broad, not focusing on individual options. The goal is to provide a broad set of options within the architecture and technology spaces which may meet the stated objectives by the customer. In Chapter 3.3.1, various viewpoints from the DoD Architecture Framework were selected to aid in visualization of the system of systems decomposition, specifically, the operational resource flow, OV-2, the operational activity decomposition, OV-5a, and the physical systems decomposition, SV-1. An example of OV-2 and OV5a within the space architecture domain is the bat chart, a graphical representation of the major physical vehicle elements and their relative location throughout the mission.

In addition to OV-2 and OV-5a, used to represent high-level operations of an architecture, techniques such as morphological analysis are used to break down the physical subsystems and the available options which may be combined to form architectures. Morph matrices, the typical result of morphological analysis, are a good example of SV-1. The combination of OV-2, OV-5a, and SV-1 are used to fully populate the architecture space and technology space defined by STSD.

### 5.2.3 Step 3: Identify Modeling and Simulation Environment(s)

The third step of IntegrATE is to identify the modeling and simulation environment(s) which may be required to evaluate the potential designs and technologies defined in the architecture and technology spaces from step 2. Because the architecture and technology spaces are defined at the subsystem level, the models collected and developed must be at the subsystem level. This will require a simulation environment capable of integrating many subsystem-level models. Industry provides such simulation environments for this purpose, such as Phoenix Integration's ModelCenter software, a graphically-based simulation environment, or the OpenMDAO project's Python module, a code-based simulation environment. The core requirement of the modeling and simulation environment(s) is to provide the ability to integrate many subsystem-level models to allow automated analysis of many different architectures and technologies simultaneously capable of translating the architecture and technology space options into the objective space figures of merit. The collection of inputs and outputs to the modeling and simulation environment(s) form the design space of the problem. Considerations of case run time and parallelization should be made during this step due to the potential for large numbers of cases which may need to be evaluated, discussed in step 5.

### 5.2.4 Step 4: Map Design and Objective Spaces

The fourth step of IntegrATE is to map the various spaces developed during steps 1-3. The mapping concept of the various spaces was discussed in Chapter 3.3.1. These mappings are a form of transformation between the various spaces. They are typically subjective in nature, defined by a subject matter expert, technologies, architect, etc. Each option in the architecture and technology spaces is mapped to at least one of the input metrics in the design space. These options may be mapped to more than one design space attribute. This implies that the design space attributes may be functions of multiple architecture and technology options. The output metrics of the design space are then mapped to the figures of merit in the objective space. The technology space options may also map to the objective space as well. Combinations of design space outputs may be combined to form a single objective space figure of merit. Unmapped design space inputs will need to have assumptions applied to create default inputs. Not all outputs from the design space must be mapped to the objective space. However, these unmapped design space metrics are not required and represent losses in performance of the analysis environment. This may warrant revisiting step 3 to further refine the modeling and simulation environment(s) to perform analysis more efficiently since the number of alternatives, and hence computational requirements, are typically of concern. Organizational techniques, such as an N-squared diagram, may be helpful in organizing and mapping inputs and outputs among the various models within the modeling and simulation environment.

### 5.2.5 Step 5: Evaluate Cases

The fifth step of IntegrATE is to perform evaluation of the design alternatives and technologies described by the combined architecture and technology spaces. Typically, the number of alternatives to be evaluated will be substantial due to the discrete, categorical nature of system of system architectures and technologies requiring full factorial DOEs. Also, issues of combinatorial explosion which frequently occur when designing complex systems of systems further exacerbate the issue of large numbers of alternatives. Further research and development is required regarding this step to better handle the large numbers of alternatives that typically exist. Currently, if the number of alternatives of the full factorial DOE is prohibitively large, it may be necessary to return to step 2 and further scope the architecture and technology spaces to reach a number of alternatives the modeling and simulation environment is able to handle. There is no fixed number of alternatives that can be suggested due to variability in computational effort required by the modeling and simulation environment. For instance, environments with run times of less than one second per design will be capable of analyzing many more designs for a fixed amount of computational capability compared to environments with run times of minutes. Obviously, increasing the computational capability allows for a greater number of designs to be run in a fixed amount of time. This may prompt high levels of parallelization of the modeling and simulation environment(s) to allow for evaluation of the most cases.

### 5.2.6 Step 6: Explore Results

The final step of IntegrATE is to explore the results and make final decisions. The primary goal of this step is to explore the objective space for trends and interactions which otherwise may have been overlooked by traditional methods. The individual observation of very large sets of alternatives simultaneously may be limited by the capability of visualization software and techniques. However, observation of individual alternatives may provide useful information relating to the frequency of certain high-level design decisions which meet a specific criteria, or the distribution of optimal designs based on specific technologies. Observation of the results as portfolios has the potential to lose information about individual architectures. Optimal alternatives may be obscured in portfolios deemed suboptimal. However, exploration of the results should not be limited to individual alternatives.

Large numbers of alternatives may be grouped together in portfolios to better observe trends and relationships in the objective space due to the options within the architecture and technology spaces. Traditionally, methods choose a relatively narrow and fixed set of grouping criteria to form these portfolios. For instance, a technology method may choose to only observe results in the form of technology portfolios, or an architecture design method may focus on physical architecture options on which alternatives are grouped into portfolios. Because IntegrATE performs both architecture design and technology evaluations simultaneously, no single grouping criteria is suited to analyzing the results. Rather, various grouping criteria should be evaluated to explore the objective space in an attempt to reveal trends due to the combined architecture and technology spaces. This also provides the ability to study interactions between the technology and architecture spaces unachievable by traditional methods due to the subsystem-level nature of IntegrATE. For instance, a technology's ability to shift a subset of the architecture space within the objective space may be observed, or relationships between seemingly unrelated technologies and architecture subsystems may be revealed and observed. Exploring results in this manner requires summarizing portfolios of alternatives in the form of distributions of the figures of merit for the portfolio.

Typically, solutions to a customer's original need resulting in the design of new products culminates in some final decision with regard to the original customer requirements. However, the highly subjective nature of conceptual design results in large variations in the "best" design or family of designs as a solution. The customer weightings on the specific figures of merit in the objective space are extremely subjective and highly influential in determining a final result. There exist many structured techniques for the purpose of decision making: technique for order preferencing by similarity to ideal solution (TOPSIS), technology frontier, and resource allocation are a few. IntegrATE is flexible enough to allow a variety of decision making techniques to be utilized which may be geared towards a specific problem.

## 5.3 Implementation: IntegrATE Framework Proof of Concept

Manned Mars missions have been studied since the mid 20th century. In the early 21st century, manned missions to Mars have garnered increased interest as increasing capabilities have developed over the decades. The Integrated Architecture and Technology Exploration (IntegrATE) framework was applied to a 2033 crewed Mars flyby study as a proof of concept. This study is a good benchmark for the IntegrATE framework due to the large variability in potential architectures for performing a manned Mars flyby as well as the substantial number of new technologies which must be developed to achieve the objective.

### 5.3.1 Step 1: Define the Problem

Since the end of the United States Apollo Program, manned space flight has been limited to low earth orbit operation. With NASA's retirement of the aging Space Shuttle fleet, the Space Launch System (SLS) is being developed to enable human exploration of deep space. Through the development and operation of various space laboratories such as Skylab, Mir, the International Space Station, and Tiangong, humans have begun to research and develop the technologies required for manned, long-duration, deep-space missions. These technologies, coupled with the new capabilities provided by the SLS, provide the basic requirements to begin moving human presence beyond low earth orbit. However, to achieve the goal of manned Mars exploration, further development is required.

Before sending humans to the surface of Mars, a manned Mars flyby mission is proposed to further refine and develop technologies and capabilities required for a full manned surface mission to Mars. Based on the development of the SLS, the 2033 launch window for Mars allows ample time to develop the basic systems required to send humans on a flyby mission to Mars. However, the specific forms of many of these systems are still in the conceptual design phases with many potential options. One such component is the in-space transportation vehicle. There are many designs which may provide the necessary capabilities to send humans to Mars. The goal of this proof of concept study is to explore the potential in-space transportation options for a manned Mars flyby in 2033.

#### *5.3.1.1 Objective Space*

Typically, the four primary objective categories for evaluating space architectures are performance, cost, risk, and schedule. For this study, performance, cost, and risk are of primary importance. The figures of merit on which to evaluate alternatives will be total number of launches, architecture mass, gross vehicle propellant mass fraction, gross vehicle cost, and technological complexity. These figures of merit define a five-dimensional objective space, shown in Table 23. Because each launch vehicle has an inherent cost associated with it which is typically very high, reducing this metric is most favored and given the greatest weighting in the objective space. Additionally, the cost of developing and manufacturing the transportation elements themselves is of interest. Total architecture mass is of little importance on its own because the total number of launches is a function of this metric. However, given two architectures which have identical values for all other metrics, the lowest mass alternative will be selected. A metric for defining the structural efficiency of the architecture is provided in the form of the propellant mass fraction of the architecture. Higher PMF values indicate a more structurally efficient design. Both mass and PMF will be considered such that the effects of technologies on both the physical structure, as well as the propellant loads of an alternative may be observed. Finally, technological complexity is a direct function of the number of active technologies applied from the technology space.

**Table 23:** Proof of Concept Objective Space

| **Objective Metric** | **Weight** | **Target** | **Units** |
|---|---|---|---|
| Number of Launches | 0.4 | Minimize | |
| Architecture Gross Mass | 0.1 | Minimize | MT |
| Gross Vehicle PMF | 0.1 | Maximize | |
| Gross Vehicle Cost | 0.2 | Minimize | MYr |
| Technological Complexity | 0.2 | Minimize | |

### 5.3.2 Step 2: Define Architecture and Technology Spaces

As described by the IntegrATE framework, the trade space for this notional problem consists of all options from the architecture and technology spaces. The architecture space is broken into two primary sets consisting of the vehicle options and the mission options. The technology space will be a set of technologies which interact with specific vehicle and mission options. The following sections describe these options for the proof of concept problem being explored.

#### *5.3.2.1 Architecture Space*

Within the architecture space are all options pertaining to both the vehicle and mission. The mission is assumed fixed to a single 2033 Mars flyby trajectory and CONOPs. This fixed mission and CONOPs are graphically represented by the bat chart shown in Figure 75. The series of mission events consists of a buildup of elements in a lunar distant retrograde orbit (LDRO), followed by a transfer to a lunar distant high Earth orbit (LDHEO) where the crew will launch from earth and rendezvous before performing a trans-Mars injection burn of 629 meters per second. The transfer from cis-lunar space to LDHEO shall take 200 days. Transit to Mars assumes a 262 day duration and 40 meters per second of correction maneuvers. A single powered fly burn of 1,290 meters per second is performed at Mars to achieve an Earth intercept trajectory. Transit back to Earth is assumed to require 318 days with 40 meters per second of correction maneuvers. Upon reaching Earth, the transport vehicle will perform an Earth orbit insertion burn of 1,072 meters per second to place the transit habitat into LDHEO. The crew returns to Earth and the transit habitat is transferred into an LDRO for storage. Disposal of any vehicle elements along the mission assumes a five meter per second burn.

In order to allow a more robust set of space transportation vehicle options within the architecture space to be studied, it is assumed that the launch vehicle for placing the required elements into orbit will be the SLS with a payload capability of 54 metric tons to cis-lunar trajectories. Any crew transport to and from Earth is performed by Orion and SLS. The Mars transit habitat is assumed to be a fixed design with a dry mass of 20 metric tons (MT) along with 12 MT of logistics.

The black boxes in Figure 75 represent transportation vehicle to be designed. The vehicle options are limited to no more than three propulsive stages. The main propulsion systems for these stages are limited to either liquid bipropellant engines or nuclear engines. Other vehicle options consist of propellant pressurization, propellant types for both the main propulsion system and reaction control system, tank configuration, power generation, and passive thermal control multi-layer insulation thickness. These options are summarized in Table 24. 162 unique vehicle stage alternatives exist after considering incompatibilities within the stage options.

Transportation vehicles shall be limited to only utilizing common stages with

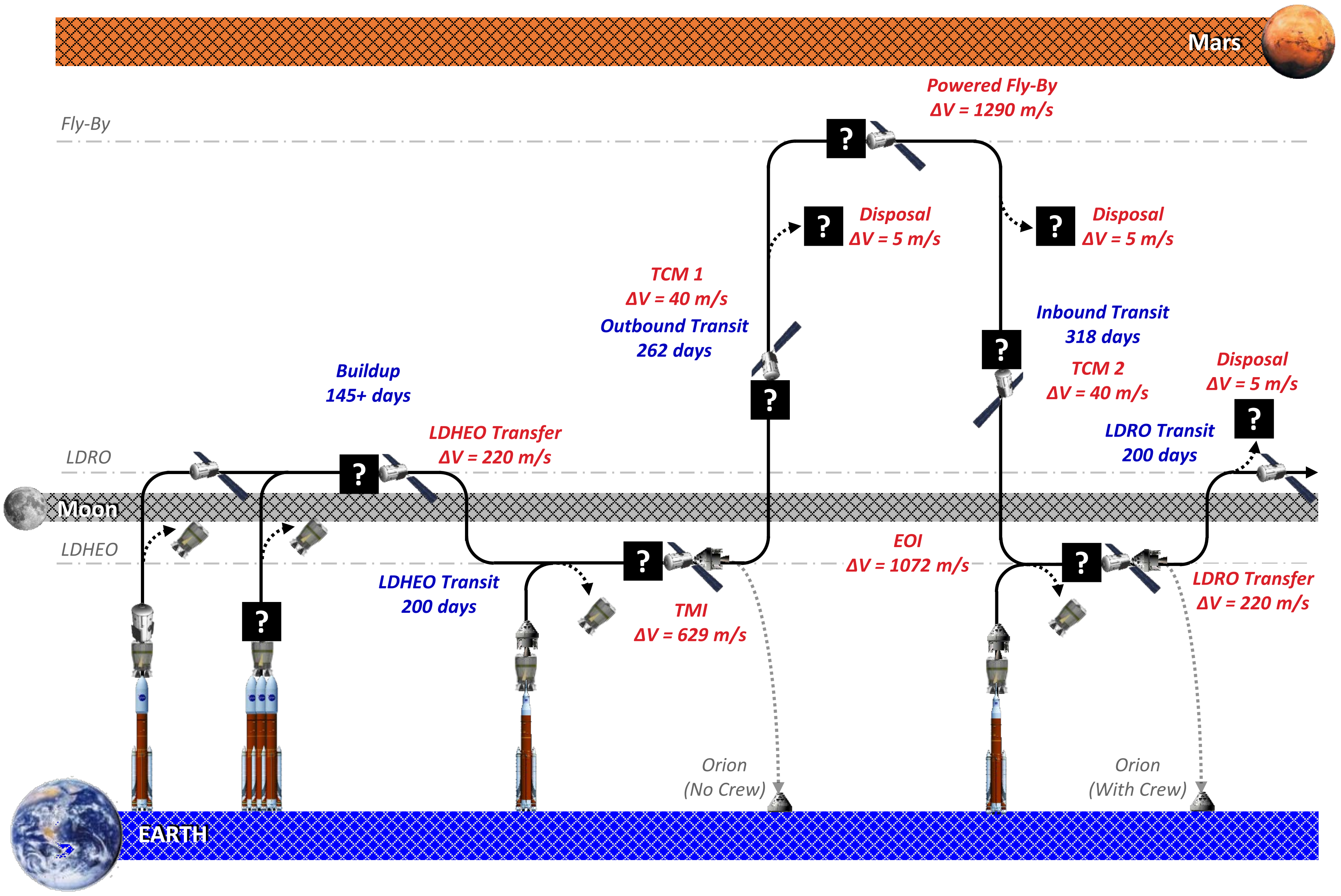


**Figure 75:** Mars Fly-By Architecture Bat Chart

**Table 24:** Vehicle Subspace Options

| **Category** | **Options** | | | | |
|---|---|---|---|---|---|
| **Vehicle** | | | | | |
| Number of Stages | 1 | 2 | 3 | | |
| **Stage(s)** | | | | | |
| MPS Class | Liquid | Nuclear | | | |
| MPS Propellant | $LO_2/LH_2$ | $LO_2/LCH_4$ | NTO/MMH | $LH_2$ | $N_2H_4$ |
| RCS Propellant | $LO_2/LCH_4$ | NTO/MMH | $N_2H_4$ | | |
| Tank Configuration | Stacked | Disk | Single | | |
| Power System | Solar | RTG | | | |
| MLI Layers | 10 | 30 | 50 | | |

identical stage options. This results in a total of 486 vehicle alternatives. Due to the fixed mission along with the assumptions provided earlier, the architecture space consist of only these 486 vehicle alternatives. Finally, because schedule is of little importance in this study, it will be assumed that development of the transportation elements is complete by the 2033 flyby date. No analysis shall be performed on development time of vehicle elements.

#### *5.3.2.2 Technology Space*

The technology space shall consist of eight technologies across six vehicle subsystems: engines, tanks, structures, power, thermal, and avionics. The engines subsystem technology is a low leak valve technology which limits propellant leak during engine start and stop. The tanks subsystem technologies are composite materials for propellant storage to minimize tank mass, integrated propellant storage systems for the main propulsion system and reaction control system to minimize the need for additional tanks for the reaction control system, and autogenous pressurization to reduce pressurant masses. The structures subsystem consists of a composite material technology to reduce the overall mass of the structures subsystem. The power subsystem technology is high capacity energy storage devices which reduce the size and mass of batteries in the power subsystem. The thermal subsystem technology is active cryocooling which mitigates propellant loss at the cost of additional hardware mass and power requirements. Finally, the avionics subsystem technology is wireless sensors which reduce the need for physical cabling, reducing the overall mass of the subsystem. These technologies are summarized in Table 25.

**Table 25:** Proof of Concept Technology Space Options

| **Category** | **Options** | |
|---|---|---|
| Wireless Sensors | TRUE | FALSE |
| Low Leak Valves | TRUE | FALSE |
| High Capacity Energy Storage | TRUE | FALSE |
| Composite Structures | TRUE | FALSE |
| Composite Propellant Tanks | TRUE | FALSE |
| Integrated MPS/RCS Propellant Storage | TRUE | FALSE |
| Autogenous Pressurization | TRUE | FALSE |
| Active Cryocooling | TRUE | FALSE |

Each technology has only two options for its activation state on the architecture, either True or False. The performance of a given technology is considered to be static, meaning there is no variability in the performance of a given technology. Similar to the vehicle options, because schedule is of little importance in this study, it will be assumed that development of any applied technologies are complete by the 2033 flyby date. No analysis shall be performed on development time of technologies. The assumptions made throughout the architecture and technology spaces will maintain a manageable number of total alternatives to be evaluated in this proof of concept study.

### 5.3.3 Step 3: Identify Modeling and Simulation Environment(s)

As was discussed in Chapter 4, no tool exists for the purpose of performing subsystem-level analysis of architectures and technologies in the space transportation domain. As such, the developed tool, DYREQT, and the various models described throughout Chapter 4 shall be utilized as the modeling and simulation environment for this proof of concept. Cost values will be estimated based on the implementation of the TransCost 7.1 model discussed in Section 4.5.3.6. To reiterate, this cost model is a historical mass-based costing model. Both development and production cost are estimated based on mass and specific architecture options. The infusion of technologies is also included in the costing of the design. However, all technologies incur the same cost penalty. Additionally, the time, cost, and risk associated with infusing multiple technologies simultaneously is not considered. The resulting digital test bed utilized for this proof of concept is of similar form as that developed for the experimentation of this dissertation, described in Section 5.1.1. The primary differences between the two are in the definition of a scoped architecture space for this proof of concept, along with a more broad set of technology combinations evaluated, and a different set of figures of merit in the objective space.

#### *5.3.3.1 Design Space*

Sizing is performed over six vehicle subsystems: engines, tanks, structures, power, thermal, and avionics. These six subsystem models are integrated with DYREQT to provide mass, power, and heat load estimates for each subsystem of each stage, as well as total burnout and propellant masses for each stage. Cost data is estimated using the TransCost 7.1 historical weights-based cost estimation model. The cost model utilizes outputs from DYREQT to estimate the development, production, and gross cost of each stage and the entire vehicle. The design space consists of the collection of outputs from DYREQT and the cost model. The tables in Appendix C provide a list and description of the various inputs to the design space.

### 5.3.4 Step 4: Map System Spaces

The modeling environment identified in step three resides within the design space. The architecture and technology spaces identified in step two feed the inputs required by the design space. The outputs of the design space supply metrics to the figures of merit in the objective space. Figure 76 provides a graphical overview of these mappings.

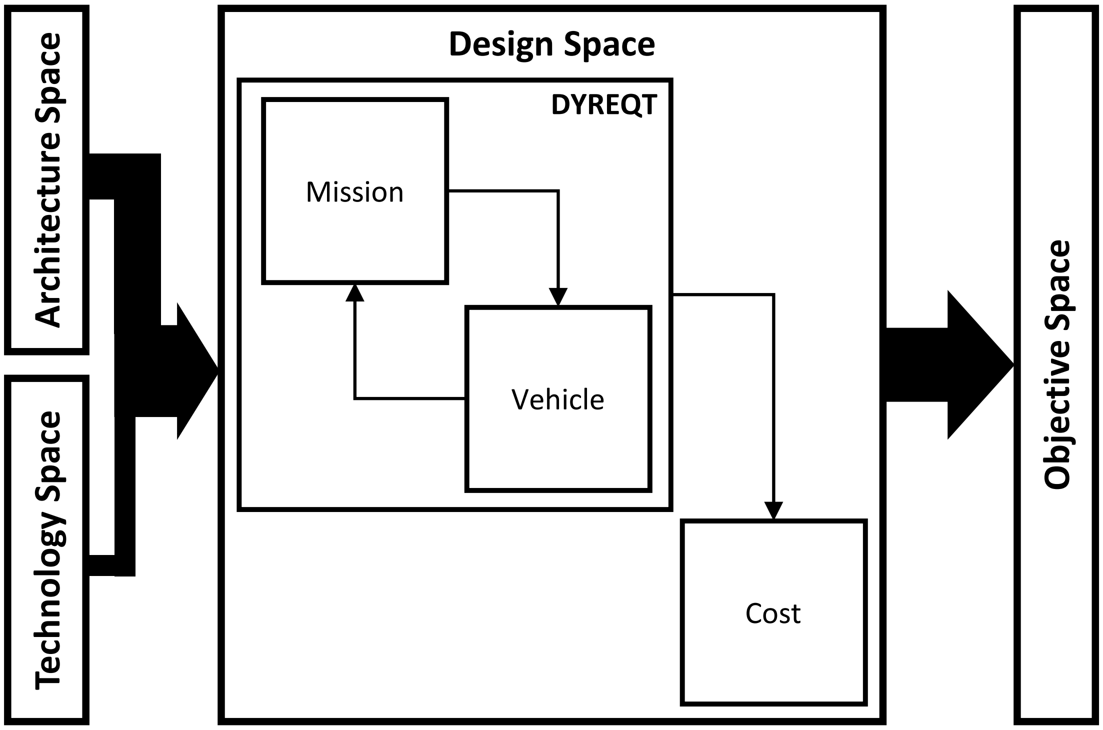


**Figure 76:** Proof of Concept Spaces Mappings

The architecture and technology space categories are mapped to specific design space parameters, shown in Table 26 and Table 27 respectively. A description of the design space parameters and default input values can be found in Appendix C. The number of vehicle stages and architecture contains has overarching effects on the number of events, event sequencing and CONOPs, as well as the total number of vehicle elements defined. For instance, a vehicle with only one stage will have fewer events related to dropping of spent stages compared to a three-stage vehicle. Similarly, the MPS class architecture category affects the mission event sequencing and element subsystem composition. Particularly, the nuclear MPS class option consists of drop tank and in line tank configurations which warrant special treatment in the

**Table 26:** Architecture Space Parameter To Design Space Attribute Mappings

| **Architecture Space Parameter** | **Design Space Attribute(s)** |
|---|---|
| **Vehicle** | |
| Number of Stages | event_list, element_list |
| **Stage(s)** | |
| MPS Class | event_list, start_penalty_mps, total_thrust_mps, engine_thrust_mps |
| MPS Propellant | isp_mps, mixture_ratio_mps |
| RCS Propellant | isp_rcs, mixture_ratio_rcs |
| Tank Configuration | num_fuel_tanks_mps, num_ox_tanks_mps |
| Power System | generator_type |
| MLI Layers | mli_layers_mps, mli_layers_rcs |

**Table 27:** Technology Space Parameter To Design Space Attribute Mappings

| **Technology Space Parameter** | **Design Space Attribute(s)** |
|---|---|
| Wireless Sensors | wireless_sensors |
| Low Leak Valves | start_penalty_mps, start_penalty_rcs |
| High Capacity Energy Storage | storage_specific_energy |
| Composite Structures | composite |
| Composite Propellant Tanks | composite_fuel_tanks_mps, composite_ox_tanks_mps, composite_fuel_tanks_rcs, composite_ox_tanks_rcs |
| Integrated MPS/RCS Propellant Storage | ivfm |
| Autogenous Pressurization | pressurant |
| Active Cryocooling | active_cooling_mps, active_cooling_rcs |

mission event sequencing and CONOPs. For the purposes of maintaining a manageable number of total cases to be evaluated, fixed values were assigned to the design space parameters for each of the architecture and technology space options. However, the values do not have to be fixed and may take on continuous ranges where applicable. For instance, to study the effects of technologies and subsystem performance, architecture and technology space options may be mapped to continuous design space parameters to account for performance uncertainties. Appendix D.3 and Appendix D.4 are the default mission and vehicle inputs to DYREQT within the design space.

Table 28: Objective-Design Space Mappings

| Objective Space Metric | Design Space Attribute(s) |
|---|---|
| Number of Launches | num_stages, total_payload, element(#)_gross_mass, element(#)_burnout_mass |
| Architecture Gross Mass | vehicle_gross_mass |
| Gross Vehicle PMF | element(#)_gross_mass, element(#)_propellant_mass_mps, element(#)_propellant_mass_rcs |
| Gross Vehicle Cost | vehicle_gross_cost |
| Technological Complexity | element0_mps_start_penalty, element0_storage_specific_energy, element0_composites, element0_composite_fuel_tanks_mps, element0_ivfm, element0_pressurant, element0_active_cooling_mps, element0_wireless_sensors |

For unmapped design space parameters, these values are the defaults utilized during analysis. For mapped design space attributes, values are determined based on the options selected from the architecture and technology spaces. The values associated with these options can be found in Table 33 through Table 53 in Appendix C.

The objective space metrics are mapped to at least one of the design space parameters. Table 28 shows the mapping of each objective space metric to the design space. Entries with the # symbol denote multiple stage elements dependent on the selections in the architecture space defining the number of vehicle stages. For the number of launches objective space metric, assumptions regarding the launch vehicle capability are used along with the design space parameters to estimate the objective value. Technology complexity is a factor dependent on the number of active technologies. It is assumed that when a technology is active, it is applied everywhere on the vehicle where applicable. Because all architecture alternatives contain at least one stage, only the first stage design space parameters are utilized in determining the technology complexity metric. Technological complexity for this study is defined as an integer representing the total number of technologies utilized by a given design alternative. The higher the number of utilized technologies, the higher the technological complexity.

### 5.3.5 Step 5: Evaluate Cases

Because the architecture and technology space options are mapped to fixed values of the design space parameters, a full factorial DOE is performed over the combined architecture and technology space with no need for additional DOEs due to continuous architecture and technology space options. This results in a total of 486 architectures and 256 different sets of technologies which may be applied to those architectures, resulting in a total of 124,416 alternatives. Utilizing an Intel® Core$^{TM}$ i7-4810MQ at 2.80 GHz, the average case evaluation time was on the order of 1.5 seconds requiring 2.16 days of processing time. Utilizing seven processing threads resulted in about 7.5 hours of wall clock analysis time. These statistics show how parallelization of the modeling and simulation environment can drastically improve the throughput of alternatives as the total number of alternatives grows due to combinatorial explosion. A modeling and simulation environment capable of parallelization, such as DYREQT, coupled with access to high-performance computing clusters, results in the ability to evaluate very large alternative spaces in the relatively short periods of time typically available during the conceptual design phases.

### 5.3.6 Step 6: Analyze Results

The initial set of alternatives consisted of a mostly even mix of alternatives grouped by main propulsion system propellant type. Liquid hydrogen monopropellant-based alternatives were only compatible with nuclear-based alternatives resulting in its under representation in the objective spaces, as seen in Figure 77. The analysis results were filtered to limit total architecture mass to less than 200,000 kg to remove outlier designs which skew distributions during analysis. Also, designs which contain elements which are not capable of being launched on the 54 MT launch vehicle limit were omitted from the final analysis of results. These criteria reduced the total number of alternatives from 124,416 to 95,127. Figure 78 shows the resulting distributions of invalid alternatives by main propulsion system propellant type, while Figure 79 shows the valid alternatives distribution. These two figures show the disproportionate level of cryogenic propellant based alternatives which are deemed invalid for this study. For liquid bipropellant alternatives, this is likely due to very large propellant mass requirements as a result of propellant boiloff during the 1125 day long mission. $LH_2$-based alternatives represent a large portion of invalid alternatives despite their relatively low representation in the original objective space due to both propellant boiloff and increased burnout mass of nuclear propulsion systems utilizing this propellant type.

Making a final decision is typically aided by the implementation of formal decision making techniques. The IntegrATE framework provides no single recommendation for choosing a decision making technique. Due to its simplicity, TOPSIS was employed to evaluate the multi-objective space utilizing customer weights identified in step one. Only alternatives which exist on the Pareto front of the five-dimensional objective space were evaluated using TOPSIS. The distribution of architecture alternatives by main propulsion system propellant type are shown by Figure 80. Liquid oxygen/liquid methane propellant based architectures make up a significant portion of the Pareto front. This is a result of the increased propulsion performance of engines utilizing this propellant compared to storable options, while having better properties for long duration storage compared to liquid hydrogen based architectures. Despite this majority, results from the TOPSIS analysis yield a single stage liquid storable architecture as the highest ranked design among the 197 alternatives which exist on the Pareto front. The alternative has a very low technological complexity. This coupled with a low stage count and non-cryogenic propulsion results in a very low cost-to-performance ratio making the alternative highly attractive for the objective weights stated during problem formulation. However, the weighting values can be adjusted to observe how the top ranked alternative changes. For instance, if there is a lower emphasis on minimizing the technological complexity and cost, three-stage nuclear-based alternatives become highly attractive due to their very high performance compared to liquid bipropellant architectures. Alternatively, liquid cryogenic bipropellant based architectures become particularly attractive when increased performance is desired while limiting the technological complexity.

For the 197 alternatives which exist on the Pareto front of the objective space, the utilization of the eight technologies in the technology space is shown by Figure 81. Autogenous pressurization is utilized by nearly one quarter of all alternatives on the Pareto front, making it the most utilized technology among these designs. Composite tanks and structures are also highly utilized among the alternatives on the Pareto front. However, active cryocooling has a relatively low utilization given the fact that the majority of alternatives on the Pareto front are cryogenic propellant based designs. This is likely a result of the boiloff rates being estimated in the model. The estimated propellant loss rate may be low enough that the total mass savings resulting from active cryocooling are not as significant as the mass savings by eliminating the propellant tank pressurization components in autogenous pressurization. Active cryocooling technologies typically have a large impact on other subsystems such as power, resulting in mass growth which offsets the propellant mass savings. Conversely, autogenous pressurization does not incur a large growth in subsystem masses, potentially making it a more advantageous technology to pursue.

However, IntegrATE enables greater exploration of the objective space compared to traditional architecture analysis and technology evaluation techniques and methods by integrating technology evaluation and architecture analysis at subsystem levels.

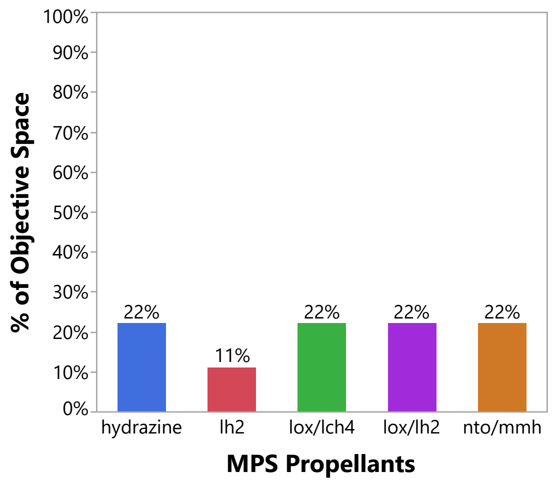


**Figure 77:** Proof of Concept Problem Full Objective Space Distribution by Main Propulsion System Propellant Type

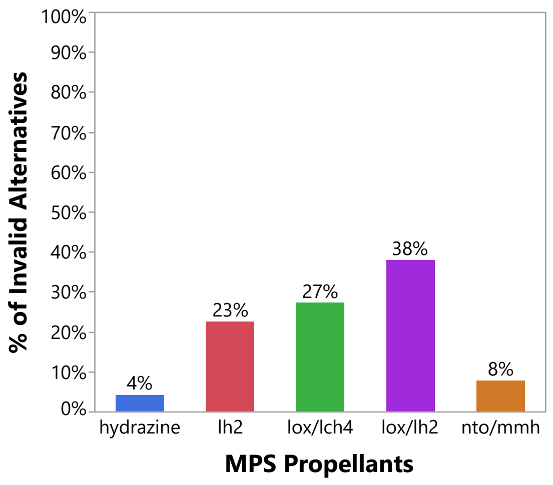


**Figure 78:** Proof of Concept Problem Invalid Alternatives Distribution by Main Propulsion System Propellant Type

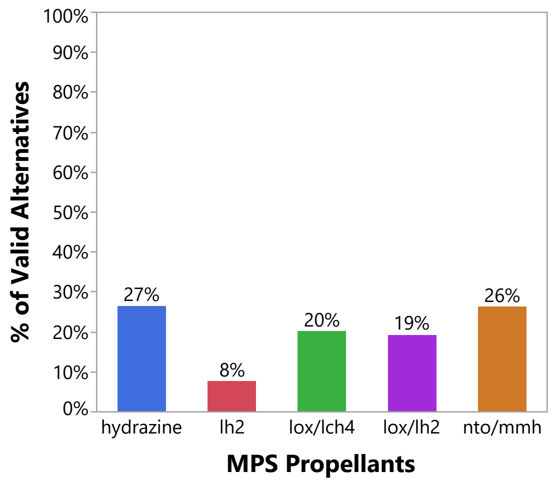


**Figure 79:** Proof of Concept Problem Valid Alternatives Distribution by Main Propulsion System Propellant Type

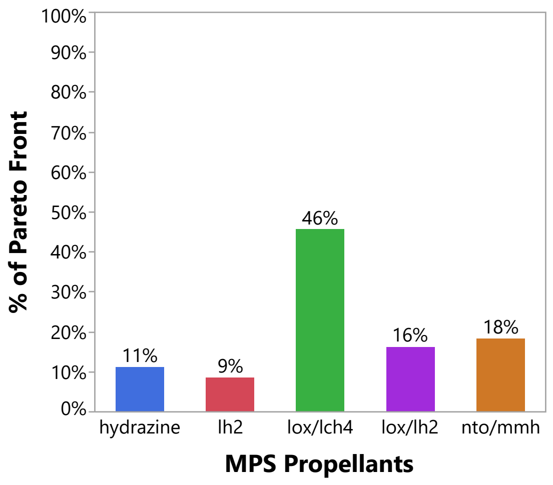


**Figure 80:** Proof of Concept Problem Pareto Front Alternatives Distribution by Main Propulsion System Propellant Type

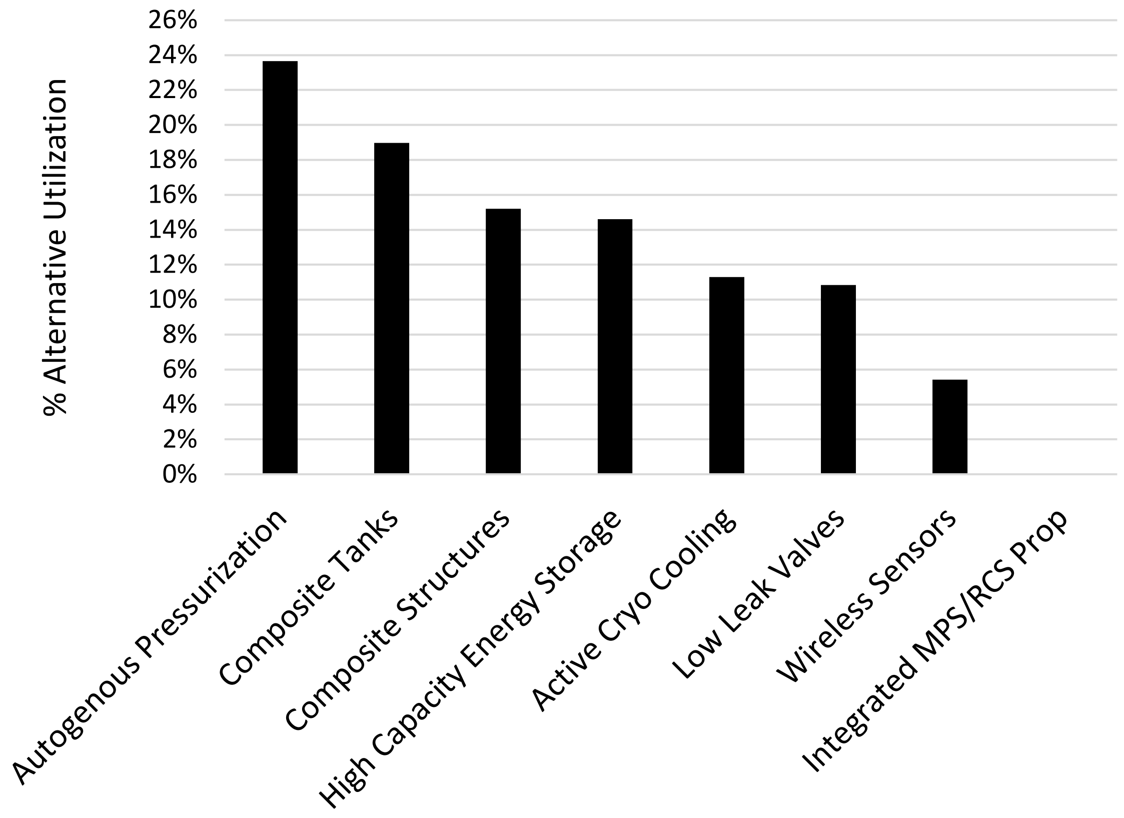


**Figure 81:** Proof of Concept Problem Technology Utilization on the Pareto Front

When the architecture space is defined at the subsystem level, it allows high-level objectives to be viewed in various ways allowing new information to be obtained linking these high-level objectives to subsystem options. For instance, when observing the distribution of architecture cost over the entire objective space, it is difficult to reach conclusions about how cost is affected by various criteria. However, grouping the objective space by both the basic propellant type and number of vehicle stages, trends in cost become immediately evident. Figure 5.3.6 shows the distributions of architecture cost for the entire objective space on the left, as well as six groups of alternatives by grouping them by either storable or cryogenic propellant options and the number of stages in the vehicle. Just observing the overall distribution on the left, it is evident that there are some trends driving the response of cost. By observing the distributions for the six groups, one can see how these options shift the cost of alternatives to form the overall objective space distribution.

Similarly, the number of launches objective can be observed for various alternative grouping criteria. Figure 5.3.6 groups architectures by number of stages, main propulsion system class, and technological complexity. Observation of the distribution of the number of launches for these architectures reveals trends which may be useful to decision makers. For instance, the number of vehicle stages has an effect on the number of launches, but not necessarily in the way one would expect. It is generally known that increasing the number of stages may increase performance by allowing the architecture to shed excess mass during the mission resulting in reduced mass. However, a lower number of vehicle stages shows a reduction in earth to orbit number of launches for the given architecture. This is likely due to the fact that there is a greater total burnout mass of vehicle elements which may not be packaged as well in a launch vehicle. The number of launches assumes that the burnout mass of a vehicle element must fit within the 54 MT launch vehicle limit, but the propellant mass may be distributed among other launch vehicles. This allows relatively efficient packing of total architecture mass compared to having multiple vehicle elements which must be packaged in launch vehicles, as well as their propellant. Obviously, if one assumes the propellant must be preloaded in the vehicle element, then this trend will change dramatically. Nuclear based-architectures also exhibit a reduced number of launches compared to liquid-based alternatives. Though nuclear architectures typically have a much greater burnout mass compared to liquid-based alternatives, the much greater efficiency of nuclear propellant alternatives resulting in reduced propellant mass leads to overall less inert mass to position in orbit, requiring fewer Earth to orbit launch vehicles. Finally, technologies have a strong impact on reducing the overall number of launches required for an architecture. By utilizing more technologies, resulting in a high technological complexity, the gross architecture mass decreases dramatically

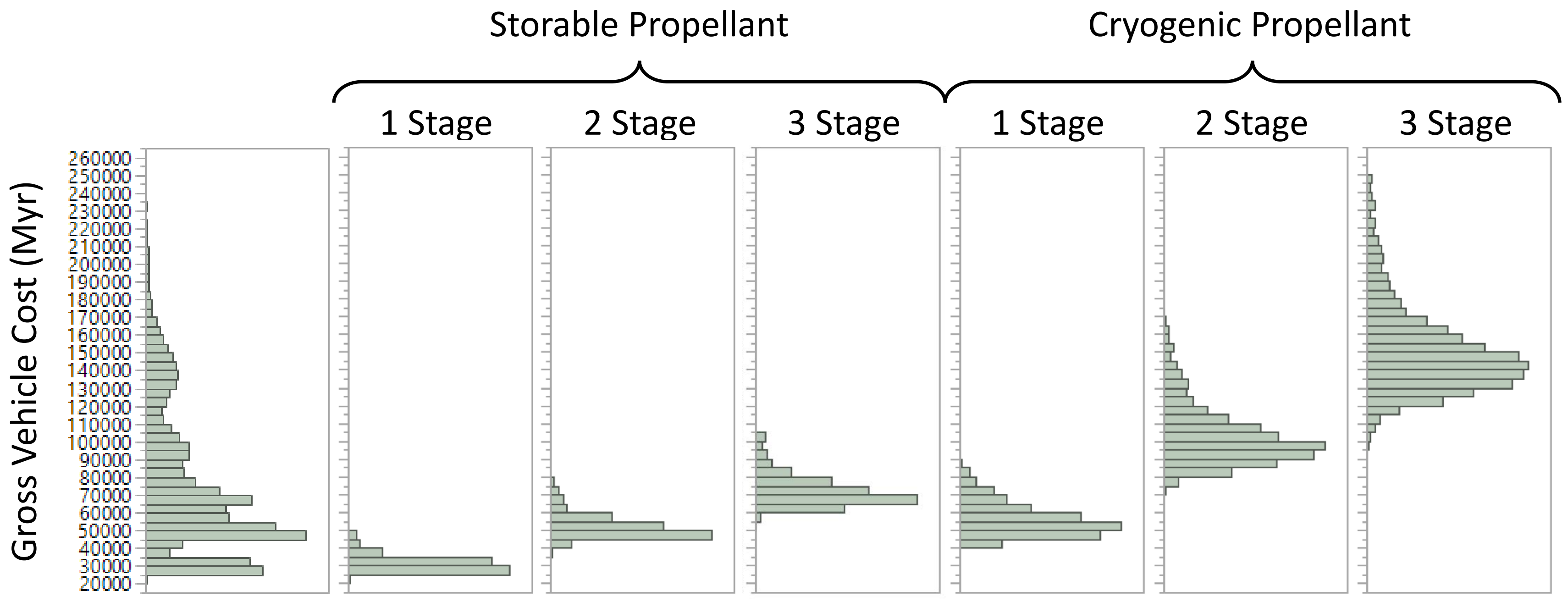
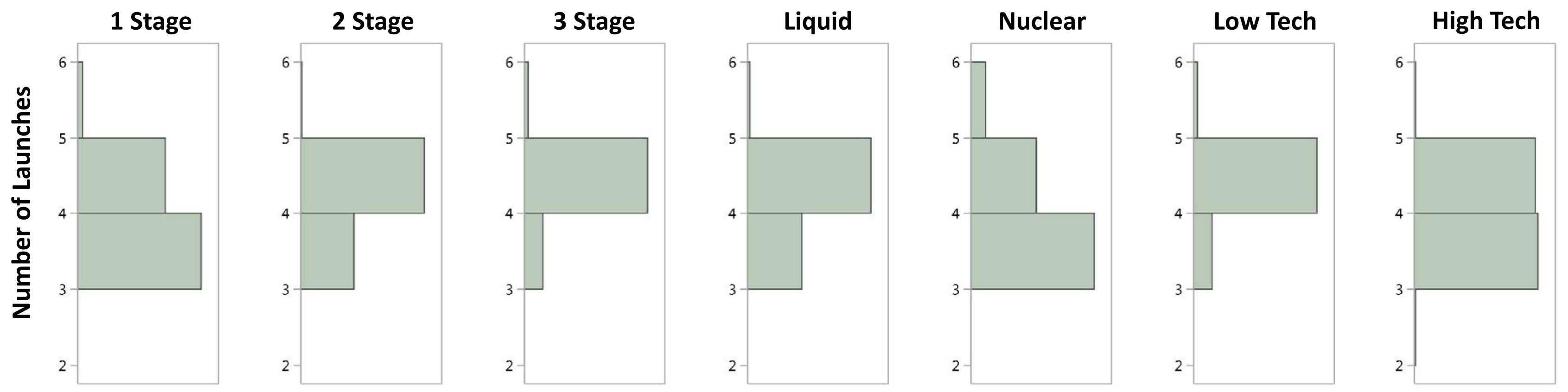


**Figure 82:** Proof of Concept Problem Architecture Cost Distributions versus Propellant Type and Number of Stages

which in turn reduces the total number of launches required. Though the distributions in Figure 5.3.6 would suggest a highly technical single stage nuclear vehicle for minimizing the total number of launches, the launch vehicle payload constraint of 54 MT would likely limit this choice, as single stage nuclear-based architectures will exhibit large burnout masses which do not fit within the 54 MT constraint.

IntegrATE enables new information to be observed with regard to how technologies impact architecture design by including technologies in the initial phases of design before down-selection of alternatives. Figure 84 shows the relationship between total architecture cost versus technological complexity. Infusing technologies has an initial negative impact on overall architecture cost. However, as technological complexity is increased by infusing more technologies, the average cost of alternatives begins to decline. This is counterintuitive to traditional thinking, but can be explained logically. By only utilizing a few technologies, the overall mass savings for a given design does not offset the cost of infusing the technologies, but as more technologies are infused, the reduction in the overall mass of the design begins to outweigh the cost of infusing additional technologies resulting in a net cost reduction. This trend is likely a result of a purely mass-based cost estimating model used in the study. Utilizing other cost models would likely result in very different observations in mean architecture cost.

Figure 85 and Figure 86 show trends which are generally expected. The mean gross mass, and as a result, the mean number of launches reduces as technological complexity increases. Both appear to exhibit relatively linear trends. Figure 87 shows the trend in mean propellant mass fraction with technological complexity. The observed trend again seems counterintuitive. Typically, higher propellant mass fractions indicate a better design with greater performance because there is less burnout mass for a given amount of propellant. However, the trends show the opposite. As the number of technologies utilized is increased, indicated by an increasing technological complexity, the mean propellant mass fraction of designs decreases. This is due to



**Quantiles**

| | | 1 Stage | 2 Stage | 3 Stage | Liquid | Nuclear | Low Tech | High Tech |
|---|---|---|---|---|---|---|---|---|
| 100% | maximum | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 0% | minimum | 3 | 3 | 2 | 3 | 2 | 3 | 2 |

**Summary Statistics**

| | 1 Stage | 2 Stage | 3 Stage | Liquid | Nuclear | Low Tech | High Tech |
|---|---|---|---|---|---|---|---|
| Mean | 3.4479857 | 3.70727 | 3.9031727 | 3.711981 | 3.4588816 | 3.8973384 | 3.4989562 |
| Std Dev | 0.5375086 | 0.4670388 | 0.3810259 | 0.4825662 | 0.6278508 | 0.3718539 | 0.5210014 |
| Std Err Mean | 0.0030938 | 0.0026055 | 0.0021035 | 0.0016283 | 0.0073505 | 0.0229295 | 0.0238052 |
| Upper 95% Mean | 3.4540497 | 3.7123768 | 3.9072957 | 3.7151724 | 3.4732906 | 3.9424879 | 3.5457319 |
| Lower 95% Mean | 3.4419216 | 3.7021632 | 3.8990498 | 3.7087895 | 3.4444726 | 3.8521889 | 3.4521805 |
| N | 30184 | 32132 | 32811 | 87831 | 7296 | 263 | 479 |

**Figure 83:** Proof of Concept Problem Number of Launches Distributions versus Select Architecture Options

two possibilities:

1. The selected technologies lead to a burnout mass growth resulting in reduced performance measured by the PMF.

2. Technologies in general reduce the overall burnout mass of architecture alternatives which in turn will result in decreased propellant masses. However, for the fixed Mars fly by mission being studied, the total propellant mass reduces faster than the burnout mass resulting in decreased vehicle propellant mass fractions overall.

However, measuring performance of an architecture or technology only by propellant mass fraction can lead to improper conclusions regarding optimal designs. Despite reduced propellant mass fraction values, technologies have a net positive impact on architectures, as indicated by reduced gross vehicle masses and number of launches.

Although the overall mean gross mass of design alternatives appears to reduce linearly with increasing technological complexity, this observation may not hold for specific groups in the architecture space. Figure 88 shows the reduction in mean gross mass versus technological complexity for two groups of vehicles, liquid-based propulsion systems and nuclear-based propulsion systems. The liquid-based alternatives follow a similar trend to that seen in Figure 85. However, nuclear-based alternatives have a highly nonlinear trend. This indicates that technologies have dramatically different impacts on different alternatives from the architecture space. Here, just a few technologies have the ability to drastically reduce the gross mass of nuclear-based alternatives. This highly nonlinear trend indicates that there are strong interactions between technologies which affect nuclear-based alternatives more than liquid-based alternatives. The fact that the liquid-based alternatives exhibit a trend similar to that seen in Figure 85 of all alternatives is likely due to the fact that liquid-based alternatives account for a much larger portion of the total number of architecture

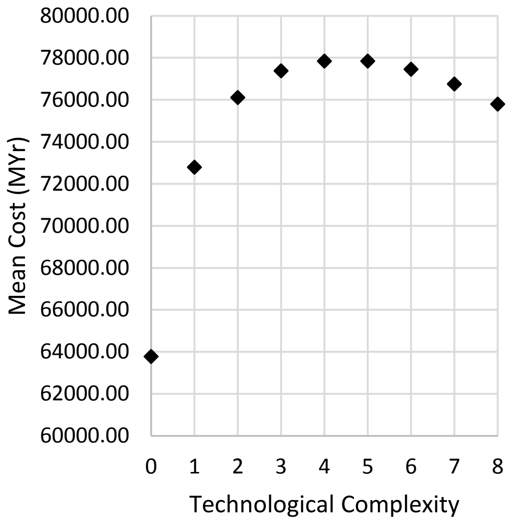


**Figure 84:** Proof of Concept Problem Mean Cost of All Alternatives with Technological Complexity

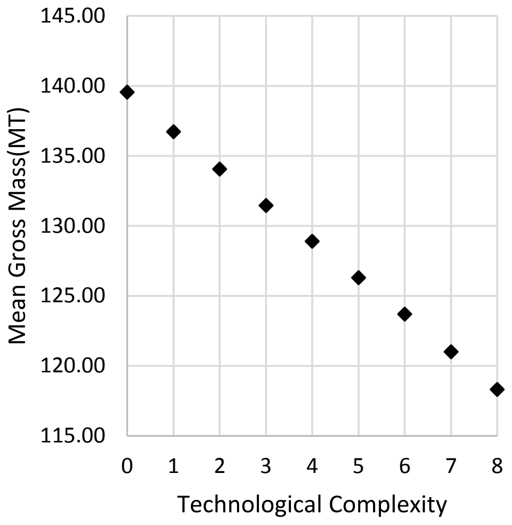


**Figure 85:** Proof of Concept Problem Mean Gross Mass of All Alternatives with Technological Complexity

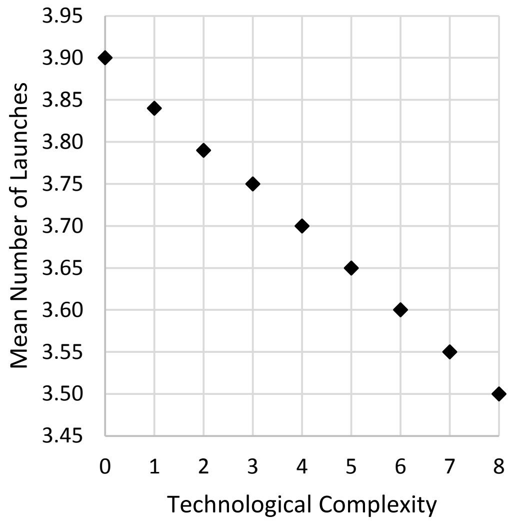


**Figure 86:** Proof of Concept Problem Mean Number of Launches of All Alternatives with Technological Complexity

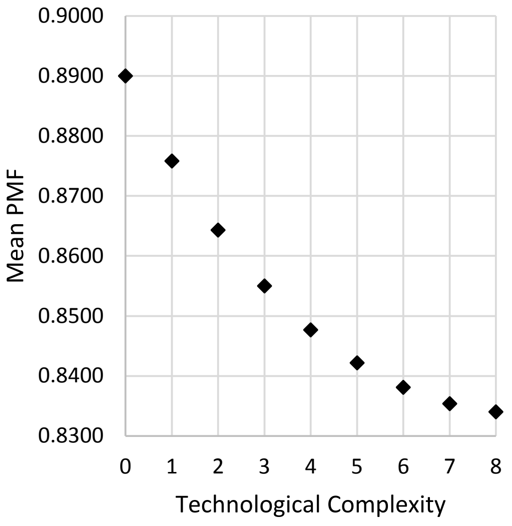


**Figure 87:** Proof of Concept Problem Mean PMF of All Alternatives with Technological Complexity

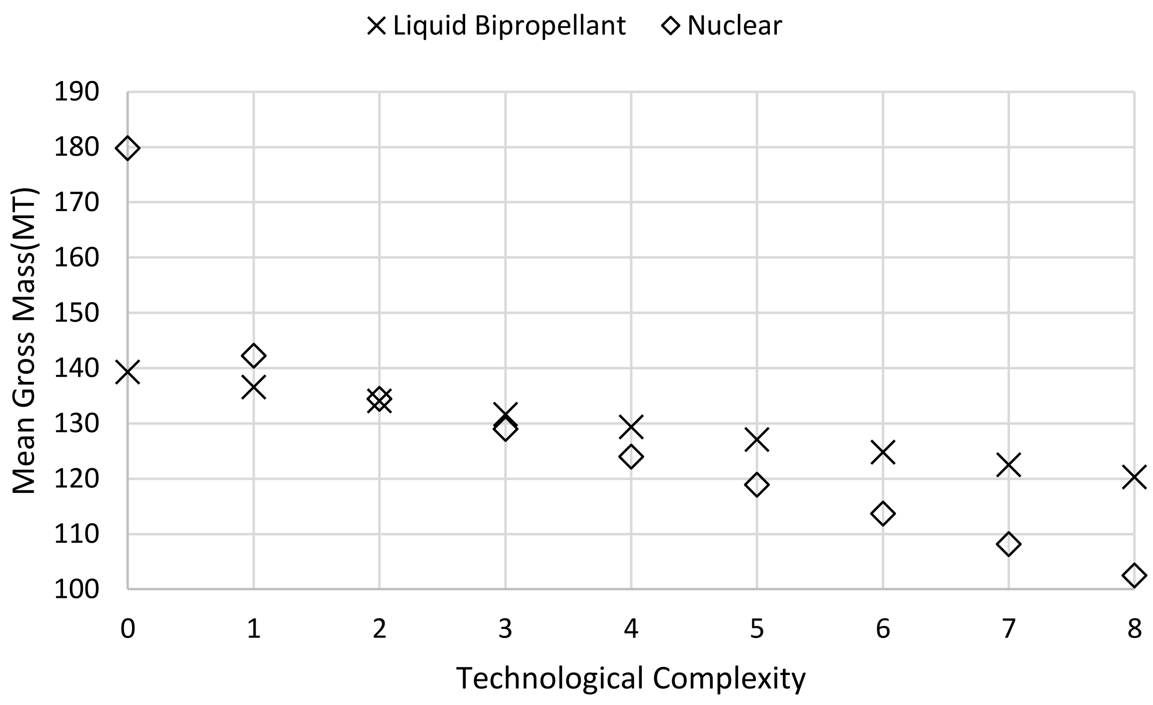


**Figure 88:** Proof of Concept Problem Variation in Mean Gross Mass with Technological Complexity of Two Distinct Vehicle Groups

alternatives.

Further observations of the performance of individual technologies can be observed across the entire set of alternatives and all objective space metrics. Table 29 shows the shift in the mean of each objective space metric between the set of alternatives utilizing no technologies and the sets of alternatives with each individual technology utilized. The first observation is with regard to active cryocooling, which has the greatest reduction in mean architecture gross mass. This technology is also associated with the highest shift in the mean cost. This is due to active cryocooling having a very large power requirement, impacting the power subsystem substantially. The total burnout mass of these designs is typically higher, resulting in increased propellant mass fractions, and because the cost model employed is mass-based, cost increases as well. However, the technology also drastically reduces propellant requirements, offsetting the increased burnout mass, resulting in a net reduction in gross mass. This is seen by the reduction in propellant mass fraction of these alternatives. Conversely, autogenous pressurization provides a much smaller performance gain compared to active cryocooling, but also results in a net cost saving compared to increased cost due to active cryocooling. The cost savings is a result of the removal of pressurization tanks in the burnout mass of these designs which is then offset by increased propellant loads, seen by an overall increase in propellant mass fractions of these alternatives. It is because of its cost savings combined with slight mass savings that autogenous pressurization was shown to be a highly-ranked technology during the initial TOPSIS analysis, while active cryocooling was ranked relatively low despite its high effectiveness at reducing mass and number of launches but at a great cost penalty. Integrated main propulsion system and reaction control system propellant was observed to be highly ineffective. It simultaneously increases cost and mass due to the addition of hardware at little to no savings by removing hardware associated with separate systems. These observations show that the performance of a technology needs to be considered at the higher architecture level. Traditionally, technologies are considered at the subsystem level, independent of the higher architecture level. Ignoring this connection would show certain technologies having negative impacts at the subsystem level, resulting in improper technology selection. Utilizing IntegrATE provides evidence for the importance of evaluating technologies at the architecture level as opposed to only the subsystem level.

**Table 29:** Shift in Objective Metric Means Due to Technologies

| | **Mean Shift** | | | |
|---|---|---|---|---|
| **Description** | **Cost (MYr)** | **Mass (kg)** | **PMF** | **Launches** |
| Low Leak Valves | -3131.84 | -266.51 | -0.0001 | -0.01 |
| High Capacity Batteries | 8174.16 | -332.31 | 0.0009 | 0.00 |
| Composite Structures | 7488.31 | -1475.84 | 0.0038 | -0.01 |
| Composite Tanks | 7886.46 | -962.96 | 0.0027 | -0.01 |
| Integrated MPS/RCS | 9469.97 | 2417.35 | -0.0078 | 0.03 |
| Autogenous Pressurization | -2522.74 | -2136.48 | 0.0043 | -0.06 |
| Active Cryocooling | 24912.64 | -12323.25 | -0.0733 | -0.23 |
| Wireless Sensors | 8508.78 | -42.97 | 0.0001 | 0.00 |

Obviously, these observations are just a select few. Certain binning criteria, such as by main propulsion system propellant type, technology complexity, and number of vehicle stages, were selected to bring out specific observations regarding these design choices. Selecting different binning criteria may lead to alternate conclusions compared to those presented in this section. IntegrATE enables a wide variety of observations to be made by integrating architecture analysis and technology evaluation at the subsystem level. It improves early knowledge with regard to the interaction between the architectures and technologies available to prevent improper design downselection during the conceptual design phases. Additionally, the level of observations is determined by the objective space metrics defined during problem formulation. This conceptual study utilized several high-level architecture design metrics such as number of launches, gross architecture mass, and gross cost. However, studies of subsystem interactions and technology effects on vehicle subsystems across many alternatives could be performed as well by defining objective metrics during problem formulation and corresponding design space outputs to evaluate the objective metrics. This study was primarily focused on high-level effects of technologies on the architecture space. Also, the effectiveness of a given technology was not studied, but is well within the capability of the IntegrATE framework. Overall, without utilizing IntegrATE, improper conclusions regarding the optimal architecture for the mission would be reached. Additionally, without utilizing IntegrATE, the effects of technologies on the architecture design would not be fully understood. Observation of reduced PMF with certain technologies would traditionally indicate a poor-performing technology. However, IntegrATE allows the causes of this trend to be observed, leading to new conclusions regarding technology performance when traced to high-level architecture objectives.

# CHAPTER VI

# CONCLUSIONS

Traditional design methods for complex systems of systems, such as the design of space transportation architectures, has historically relied heavily on subject matter experts during the early phases of design to scope the initial problem to a small set of designs which then become the basis for technology evaluation. However, ever-changing customer requirements and shifting political interests create a lack of understanding and knowledge of the design space, leading to potential cost and schedule overruns due to frequent design changes and combinatorial explosion of alternatives in the design space. These observations led to the primary research objective:

**Research Objective**

To integrate architecture analysis and technology evaluation at the subsystem level to provide a quantitative framework in an effort to increase design knowledge early in the design process.

To integrate architecture analysis and technology evaluation at the subsystem level, several questions and hypotheses were posed during a discussion of a general concept exploration process to guide the development of a new framework. However, in order to test these hypotheses, a digital test bed capable of performing integrated architecture analysis and technology evaluation at the subsystem level had to be selected. No tools were identified within the space transportation community which met this requirement. As a result, the Dynamic Rocket Equation Tool (DYREQT) and a collection of subsystem-level in-space transportation models were developed to provide a modeling and simulation environment capable of producing the necessary data for experimentation. DYREQT provides the capability to integrate user-developed subsystem models in a tool developed for space transportation architecture analysis and design.

The first research question aimed to establish the relationship between architecture and technologies as defined by this dissertation. Once established, a technique for decomposing a problem was selected which met the needs of subsystem-level integrated architectures and technologies. Research question two further guided the decomposition and definition of a problem by examining the figures of merit which should guide technology evaluations. It was determined that, due to the relationship between technologies and architectures established by research question one, similar figures of merit should be utilized for evaluating technologies.

The third research question considered the validity of the traditional paradigm of design down-selection prior to technology evaluation for systems of systems problems. Experiment one concluded that technologies can have profound impacts on specific subsets of architecture alternatives which may have otherwise been down-selected under traditional processes. These results warrant a shift in the paradigm by incorporating technology evaluation prior to architecture down-selection. However, it was recognized that this solution only exacerbates the already problematic concept of combinatorial explosion which exists in systems of systems design problems. As such, the remainder of the questions focused on how results from such large sets of alternatives should be explored.

Research question five focused on examining the effects of exploring results of architecture analysis and technology evaluation from two primary viewpoints: individual alternatives and groups, or portfolios, of alternatives. Experiment two focused on exploring the individual alternative representation of results. It was determined that very small subsets will provide inaccurate representation of the true objective space. However, subsets of the results which are still relatively small in relation to the full objective space may provide adequate representation for observing high-level design decisions. Conversely, experiment three focused on examining the grouping criteria for forming portfolios of designs and how they affect the observations in the results. The original hypothesis that portfolio-level metrics are related to portfolio size was proven incorrect, but it was shown that these portfolio metrics are more directly influenced by the options in the architecture and technology spaces. The final experiment focused on determining if grouping sets of alternatives into portfolios has the potential to obscure the true optimal design. It was shown that these optimal individual designs may become obscured within suboptimal portfolios, supporting a broad exploration of the objective space considering both individual and portfolio viewpoints. The final research question examined the trends in figures of merit utilized throughout the experiments of this dissertation to make a conjecture regarding the need for additional summary metrics when evaluating results from a portfolio viewpoint.

These questions and their results helped guide the development of IntegrATE, a new framework for the purpose of integrating architecture analysis and technology evaluation at the subsystem level. IntegrATE is an initial point of departure from traditional architecture design and technology evaluation methods aimed at increasing knowledge during the conceptual design phase through the study of interactions between architectures and technologies. The IntegrATE framework does this by allowing large numbers of architecture alternatives and technologies to be evaluated concurrently, before down-selection, to allow decisions makers to fully explore systems of systems design spaces. This does however only exacerbate the combinatorial problem already present in the design of complex systems of system. Issues such as these are why IntegrATE is classified as a framework, as opposed to a method, indicating an initial step which warrants further study and development.

To clearly demonstrate the new framework and provide solid evidence for the benefits it provides over traditional architecture analysis and technology evaluation methods, a notional manned Mars 2033 design study was performed utilizing IntegrATE and DYREQT. The study showed how various grouping criteria can help to highlight trends at the architecture level, both due to architecture design options and technologies. Additionally, it was shown that IntegrATE allows technologies to be evaluated from a high-level architecture perspective, as opposed to focusing only at the subsystem level a technology interacts with. Traditional figures of merit for evaluating the impact of individual technologies, such as PMF, were shown to provide false negative performances when viewed at the subsystem level, despite having a net positive impact on high-level architecture performance metrics which are typically used for evaluating and selecting space transportation designs.

## *6.1 Summary of Contributions*

This body of research has produced several new capabilities and advancements in the field of complex system of systems design and technology evaluation, specifically in the space architecting communities. As a result of formulating a new, integrated, approach to architecture design and technology evaluation, the IntegrATE framework was established as a point of departure from traditional system of systems design and technology evaluation methods to guide future development. Through the development of this framework, a new subsystem-level space architecture analysis tool, along with associated subsystem models, were developed.

The flexibility of the IntegrATE framework allows a wide variety of techniques to be employed such that the process of problem formulation and decision making may be tailored to the specific problem and customer requirements. The framework helps to disarm concerns regarding architecture and technology biases by providing a highly transparent framework for evaluation and exploration. Due to the high-level, integrated nature of the framework, the process will help to eliminate bias in the technologies and architectures represented within a problem, as this architect should represent a neutral party with no particular bias for specific architectures or technologies. Rather, he should be primarily interested in providing a broad representation for the problem being studied. The largest shift the IntegrATE framework makes is with regard to the traditional paradigm of design down selection before technology evaluation. Rather, IntegrATE brings technology evaluations into the conceptual design process alongside physical architecture design and analysis. The purpose for this shift is due to the potential that technologies have to dramatically shift the composition of the objective space, which under traditional methods may result in designs being overlooked as suboptimal during traditional design down selection. This is performed through the utilization of a modified set theory system decomposition technique which includes the addition of a technology space which affect design attributes through system space mappings.

Exploring results individually has the advantage of allowing clear decisions with regard to optimal designs to be made. However, by integrating architecture design and technology evaluation, the resulting objective space becomes extremely large and may be difficult to analyze. Traditional methods and techniques typically employ some form of grouping designs together to simplify the analysis of results. However, there has been little study into how these groupings may affect the results observed. Typically, groupings are formed based on the particular study being performed. During the formulation of the IntegrATE framework, multiple experiments focused on better understanding how these grouping criteria may affect the results analysis process to guide future methodology development.

In order to perform the necessary experiments to formulate the IntegrATE framework in the domain of space architecture design, new tools and models were required. Traditional modeling tools in this field typically rely on high-level models which do not allow subsystem-level trades or technologies to be introduced. Tools developed which utilize subsystem-level models are typically highly integrated and difficult to introduce user-defined models or parameters to account for the effects due to technology integration. These shortcomings led to the development of the Dynamic Rocket Equation Tool (DYREQT), a multi-discipline design, analysis, and optimization tool to aid in the integration of subsystem-level models for the purpose of space architecture design and analysis. This in turn allows subsystem-level technology integration such that technology impacts may be observed at the architecture level during the conceptual design phases.

## 6.2 Recommendations for Future Work

Frequent design changes and combinatorial explosion of the design space are two of the primary factors which lead to lack of knowledge of the design space, leading to cost and schedule overruns. This research has focused primarily on the lack of knowledge due to frequent design changes by bringing as much information about architecture design and technology evaluation into the conceptual design phases. However, this only works to exacerbate the combinatorial explosion which exists in typical system of systems problems. Additional research needs to focus on ways of handling combinatorial explosion. Historically, the most efficient way of reducing combinatorial explosion was to utilize subject matter experts to reduce the design space to a manageable region. To combat the growing uncertainties which exist in conceptual design of systems of systems, this down-selected design region is being opened up at the cost of analysis effort. Fortunately, advancements in computational capability enable large regions of the design space to be studied. However, this increase in computational capability is outweighed by the issue of combinatorial explosion as the design space is opened up. New techniques need to be developed and studied which allow the broader design space to be considered while managing the growth in combinatorial explosion in an effort to align computational effort with computational capabilities.

This challenge is out of scope for the IntegrATE framework. IntegrATE incorporates a step to scope the various trade spaces to manageable sizes before moving forward with analysis, much like current down selection processes in current methods.

Additionally, further work is required to develop the IntegrATE framework into a full methodology. This requires more study into the application of specific techniques and tools to fulfill specific tasks within the framework. Up to this point, a full study of available techniques for each phase of the framework has not been performed. Rather, single, appropriate techniques and tools which provided the essential capabilities to meet the primary research objective of this body of work were selected, with little study of competing techniques and tools. For instance, Set Theory System Decomposition was selected as a means for decomposing the problem in a way suitable for integrating technology analysis and architecture design at a subsystem level. However, other decomposition methods may be better suited to the task of decomposing these complex systems of systems. The integration of certain model-based system engineering techniques and paradigms is highly applicable to the system of systems nature posed by architecture design and technology evaluation. The integration of these techniques and their ability to clearly define and relate system of systems requirements, design, and analysis can prove valuable throughout the development of new methods based on the IntegrATE framework. The experiments in this dissertation provide insight into analyzing the complex objective spaces which result from integrated architecture analysis and technology evaluation. By observing how the objective space of subsets or groups of design are affected by such parameters as subset size and grouping criteria, new techniques and procedures may be developed for processing evaluated results. However, further research into potential data analysis techniques may aid in implementing findings from this study to better explore these complex objective spaces. Finally, no consideration was given to implementing technology uncertainty into IntegrATE. There is no limitation inherent to IntegrATE which prevents uncertainties from being considered. It was simply outside of the scope of this body of work. However, technology uncertainty is an important part of the selection process and should be an integral part of a fully-developed, integrated architecture analysis and technology evaluation methodology.

Development of the IntegrATE design framework required the development of a multi-discipline design, analysis, and optimization tool due to a gap in current analysis capabilities. The development of the Dynamic Rocket Equation Tool (DYREQT) is a valuable step forward for an end-to-end analysis capability for complex space architectures. The tool provides the ability to integrate various user-developed models in an environment specifically tailored for the space architecture design domain. However, further effort is required to integrate high fidelity trajectory modeling tools into DYREQT to provide more detailed and automated mission modeling. In addition to increasing the fidelity and integration of trajectory optimization with vehicle sizing, further research into the implementation of collaborative optimization may help to increase computational capability by increasing analysis performance for the large systems of systems problems formulated using DYREQT.

# APPENDIX A

# EXAMPLE DYREQT SETUP

*Note: the models used in this notional example are not provided in this document, as this is here for the purpose of demonstrating the basic problem input structure, DYREQT setup, and evaluation initiation. The actual calculations are irrelevant.*

```
# append DYREQT directory to path so it can be imported
import os,sys
from copy import deepcopy
sys.path.append(os.path.join(os.getcwd(),".."))
# import DYREQT classes
from Problems import DYREQTProblem
# import other packages
import sqlitedict
import numpy as np
# set numpy array printing
np.set_printoptions(threshold=np.nan)

######################
# SETUP INPUTS
######################

# vehicle input
sub0 = {'subelement_type':'PMFCom',
        'params':{'pmf':0.6,
                  'mps_isp':{'val':300.0,'units':'s'},
                  'rcs_isp':{'val':250.0,'units':'s'}}}
sub2 = {'subelement_type':'FixedStage',
        'params':{'mass':{'val':98.0,'units':'kg'},
                  'mps_isp':{'val':298.0,'units':'s'}}}
sub3 = {'subelement_type':'FixedMass',
        'params':{'mass':{'val':1000.0,'units':'kg'}}}
Booster1 = {'element_type':'Stage','subelement_list':[sub0],
            'params':{'auto_drop':False}}
Booster2 = {'element_type':'Stage','subelement_list':[deepcopy(sub0)],
            'params':{'auto_drop':False}}
Lander = {'element_type':'Stage','subelement_list':[deepcopy(sub0)],
          'params':{'auto_drop':False}}
Kick = {'element_type':'Stage','subelement_list':[sub2],
        'params':{'auto_drop':False}}
Payload1 = {'element_type':'Payload','subelement_list':[sub3],
            'params':{'auto_drop':False}}
element_list = [Booster1,Booster2,Kick,Lander,Payload1]
vehicle = {'element_list':element_list}

#  mission input
event0 = {'event_type':'Drop','params':{}}
event1 = {'event_type':'Burn',
          'params':{'dv':{'val':1500.0,'units':'m/s'}}}
event2 = {'event_type':'Drop','params':{}}
event3 = {'event_type':'Burn',
          'params':{'dv':{'val':42.5,'units':'m/s'},
          'system':'rcs'}}
event4 = {'event_type':'Burn',
          'params':{'dv':{'val':850.0,'units':'m/s'}}}
event5 = {'event_type':'Burn',
          'params':{'dv':{'val':42.5,'units':'m/s'},
          'system':'rcs'}}
event6 = {'event_type':'Drop','params':{}}
event7 = {'event_type':'Connect','params':{}}
event8 = {'event_type':'MassDelta',
          'params':{'mass_type':'prop','top_off':True}}
event9 = {'event_type':'Burn',
          'params':{'dv':{'val':1000.0,'units':'m/s'}}}
event10 = {'event_type':'Drop','params':{}}
event_list = [event0,event1,event2,event3,event4,event5,event6,
              event7,event8,event9,event10]
mission = {'event_list':event_list}

#  conops input (links mission and vehicle)
conops = [[{'active_elements':[4]}],
          [{'active_elements':[0,1]}],
          [{'active_elements':[0,1]}],
          [{'active_elements':[3]}],
          [{'active_elements':[2]}],
          [{'active_elements':[3]}],
          [{'active_elements':[2]}],
          [{'active_elements':[4]}],
          [{'active_elements':[3]}],
          [{'active_elements':[3]}],
          [{'active_elements':[4]}]]

architecture_definition = {'vehicle':vehicle,
                           'mission':mission,
                           'conops':conops}

######################
# SETUP/RUN MODEL
######################

# create problem
prob = DYREQTProblem(architecture_definition)

# set options
options={'disp':0,
         'opt_tol':1e-2,
         'fd_step':1e-4,
         'n2':0,
         'times':True,
         'file':'baseline'}

# setup problem
prob.setup_problem(options=options)

# solve problem
prob.solve_problem()

######################
# PRINT RESULTS
######################

# unpack results from database file and print to console
db = sqlitedict.SqliteDict(prob.options['file']+'.db', 'iterations')
data = list(db.items())[-1][1]
u = data['Unknowns']
p = data['Parameters']
obj = u['objective_value']
print('final objective value: {0}'.format(obj))
print('')
print('*'*79)
print('PARAMETERS')
print('*'*79)
for name,val in sorted(p.items()):
    print('{0}: {1}'.format(name,val))
print('')
print('*'*79)
print('UNKNOWNS')
print('*'*79)
for name,val in sorted(u.items()):
    print('{0}: {1}'.format(name,val))
```

# APPENDIX B

# EXAMPLE DYREQT OUTPUT

```
********************************************************************************
RUN TIME STATS
********************************************************************************
Total Run Time: 1.169 s
  Init:      0.010 s
  Setup:     0.421 s
  Solution:  0.151 s
  Cleanup:   0.587 s

final objective value: 3.3091343229661314


********************************************************************************
PARAMETERS
********************************************************************************
mission.event0.total_payload: [ 1000.     0.     0.     0.     0.     0.
     0.     0.  1000.  1000.  1000.     0.]
mission.event0.vehicle_gross_mass: [ 3309.13432297  2309.13432297  1386.82125308
   647.55862824   636.42982254   475.80853953
   467.63139727   369.63139727  1369.63139727
  1924.07827912  1369.63139727   369.63139727]
mission.event1.element0_inert_mass: [ 369.63139727  369.63139727  369.63139727
    0.           0.           0.           0.
    0.           0.           0.           0.
    0.       ]
mission.event1.element0_isp_mps: 300.0
mission.event1.element0_isp_rcs: 250.0
mission.event1.element0_mass_flowrate_mps: 33.990540432597605
mission.event1.element0_mass_flowrate_rcs: 40.78864851911713
mission.event1.element0_terminal_event: (10, 0)
mission.event1.element1_inert_mass: [ 369.63139727  369.63139727  369.63139727
    0.           0.           0.           0.
    0.           0.           0.           0.
    0.       ]
mission.event1.element1_isp_mps: 300.0
mission.event1.element1_isp_rcs: 250.0
mission.event1.element1_mass_flowrate_mps: 33.990540432597605
mission.event1.element1_mass_flowrate_rcs: 40.78864851911713
mission.event1.element1_terminal_event: (10, 0)
mission.event1.total_payload: [ 1000.     0.     0.     0.     0.     0.
     0.     0.  1000.  1000.  1000.     0.]
mission.event1.vehicle_gross_mass: [ 3309.13432297  2309.13432297  1386.82125308
   647.55862824   636.42982254   475.80853953
   467.63139727   369.63139727  1369.63139727
  1924.07827912  1369.63139727   369.63139727]
mission.event10.total_payload: [ 1000.     0.     0.     0.     0.     0.
     0.     0.  1000.  1000.  1000.     0.]
mission.event10.vehicle_gross_mass: [ 3309.13432297  2309.13432297  1386.82125308
   647.55862824   636.42982254   475.80853953
   467.63139727   369.63139727  1369.63139727
  1924.07827912  1369.63139727   369.63139727]
mission.event2.total_payload: [ 1000.     0.     0.     0.     0.     0.
     0.     0.  1000.  1000.  1000.     0.]
mission.event2.vehicle_gross_mass: [ 3309.13432297  2309.13432297  1386.82125308
   647.55862824   636.42982254   475.80853953
   467.63139727   369.63139727  1369.63139727
  1924.07827912  1369.63139727   369.63139727]
```

```
mission.event3.element3_inert_mass: [ 369.63125456  369.63125456  369.63125456
  369.63125456  369.63125456  369.63125456
  369.63125456  369.63125456  369.63125456
  369.63125456  369.63125456  369.63125456]
mission.event3.element3_isp_mps: 300.0
mission.event3.element3_isp_rcs: 250.0
mission.event3.element3_mass_flowrate_mps: 33.990540432597605
mission.event3.element3_mass_flowrate_rcs: 40.78864851911713
mission.event3.element3_terminal_event: (10, 0)
mission.event3.total_payload: [ 1000.     0.     0.     0.     0.     0.
     0.     0.  1000.  1000.  1000.     0.]
mission.event3.vehicle_gross_mass: [ 3309.13432297  2309.13432297  1386.82125308
   647.55862824   636.42982254   475.80853953
   467.63139727   369.63139727  1369.63139727
  1924.07827912  1369.63139727   369.63139727]
mission.event4.element2_inert_mass: [ 98.  98.  98.  98.  98.  98.  98.   0.   0.
   0.   0.   0.]
mission.event4.element2_isp_mps: 298.0
mission.event4.element2_isp_rcs: 1.0
mission.event4.element2_mass_flowrate_mps: 0.0003421866486503115
mission.event4.element2_mass_flowrate_rcs: 0.10197162129779283
mission.event4.element2_terminal_event: (10, 0)
mission.event4.total_payload: [ 1000.     0.     0.     0.     0.     0.
     0.     0.  1000.  1000.  1000.     0.]
mission.event4.vehicle_gross_mass: [ 3309.13432297  2309.13432297  1386.82125308
   647.55862824   636.42982254   475.80853953
   467.63139727   369.63139727  1369.63139727
  1924.07827912  1369.63139727   369.63139727]
mission.event5.element3_inert_mass: [ 369.63125456  369.63125456  369.63125456
  369.63125456  369.63125456  369.63125456
  369.63125456  369.63125456  369.63125456
  369.63125456  369.63125456  369.63125456]
mission.event5.element3_isp_mps: 300.0
mission.event5.element3_isp_rcs: 250.0
mission.event5.element3_mass_flowrate_mps: 33.990540432597605
mission.event5.element3_mass_flowrate_rcs: 40.78864851911713
mission.event5.element3_terminal_event: (10, 0)
mission.event5.total_payload: [ 1000.     0.     0.     0.     0.     0.
     0.     0.  1000.  1000.  1000.     0.]
mission.event5.vehicle_gross_mass: [ 3309.13432297  2309.13432297  1386.82125308
   647.55862824   636.42982254   475.80853953
   467.63139727   369.63139727  1369.63139727
  1924.07827912  1369.63139727   369.63139727]
mission.event6.total_payload: [ 1000.     0.     0.     0.     0.     0.
     0.     0.  1000.  1000.  1000.     0.]
mission.event6.vehicle_gross_mass: [ 3309.13432297  2309.13432297  1386.82125308
   647.55862824   636.42982254   475.80853953
   467.63139727   369.63139727  1369.63139727
  1924.07827912  1369.63139727   369.63139727]
mission.event7.total_payload: [ 1000.     0.     0.     0.     0.     0.
     0.     0.  1000.  1000.  1000.     0.]
mission.event7.vehicle_gross_mass: [ 3309.13432297  2309.13432297  1386.82125308
   647.55862824   636.42982254   475.80853953
   467.63139727   369.63139727  1369.63139727
  1924.07827912  1369.63139727   369.63139727]
mission.event8.total_payload: [ 1000.     0.     0.     0.     0.     0.
     0.     0.  1000.  1000.  1000.     0.]
mission.event8.vehicle_gross_mass: [ 3309.13432297  2309.13432297  1386.82125308
   647.55862824   636.42982254   475.80853953
   467.63139727   369.63139727  1369.63139727
  1924.07827912  1369.63139727   369.63139727]
mission.event9.element3_inert_mass: [ 369.63125456  369.63125456  369.63125456
  369.63125456  369.63125456  369.63125456
  369.63125456  369.63125456  369.63125456
  369.63125456  369.63125456  369.63125456]
mission.event9.element3_isp_mps: 300.0
mission.event9.element3_isp_rcs: 250.0
mission.event9.element3_mass_flowrate_mps: 33.990540432597605
mission.event9.element3_mass_flowrate_rcs: 40.78864851911713
mission.event9.element3_terminal_event: (10, 0)
mission.event9.total_payload: [ 1000.     0.     0.     0.     0.     0.
     0.     0.  1000.  1000.  1000.     0.]
mission.event9.vehicle_gross_mass: [ 3309.13432297  2309.13432297  1386.82125308
   647.55862824   636.42982254   475.80853953
   467.63139727   369.63139727  1369.63139727
  1924.07827912  1369.63139727   369.63139727]
optimization.total_payload: [ 1000.     0.     0.     0.     0.     0.
     0.     0.  1000.  1000.  1000.     0.]
optimization.vehicle_gross_mass: [ 3309.13432297  2309.13432297  1386.82125308
   647.55862824   636.42982254   475.80853953
   467.63139727   369.63139727  1369.63139727
  1924.07827912  1369.63139727   369.63139727]
vehicle.element0.element_utilities.element0_burn_time_mps_event1_0: 13.567190791698499
vehicle.element0.element_utilities.element0_burn_time_rcs_event1_0: 0.0
vehicle.element1.element_utilities.element1_burn_time_mps_event1_0: 13.567190791698499
vehicle.element1.element_utilities.element1_burn_time_rcs_event1_0: 0.0
vehicle.element2.element_utilities.element2_burn_time_mps_event4_0: 469396.7726970796
vehicle.element2.element_utilities.element2_burn_time_rcs_event4_0: 0.0
vehicle.element3.element_utilities.element3_burn_time_mps_event3_0: 0.0
vehicle.element3.element_utilities.element3_burn_time_mps_event5_0: 0.0
vehicle.element3.element_utilities.element3_burn_time_mps_event9_0: 16.31180055203983
vehicle.element3.element_utilities.element3_burn_time_rcs_event3_0: 0.27284071287463163
vehicle.element3.element_utilities.element3_burn_time_rcs_event5_0: 0.20047596651912505
vehicle.element3.element_utilities.element3_burn_time_rcs_event9_0: 0.0
vehicle.element3.element_utilities.element3_inert_mass_delta_event8_0: 0.0
vehicle.element3.element_utilities.element3_propellant_mass_delta_event8_0: 0.0
vehicle.element3.element_utilities.element3_top_off_event8_0: True

******************************************************************************
UNKNOWNS
******************************************************************************
element0_auto_drop: False
element0_burn_time_mps: 13.5672022002112
element0_burn_time_mps_event1_0: 13.567190791698499
element0_burn_time_rcs: 0.0
element0_burn_time_rcs_event1_0: 0.0
element0_burnout_mass: 369.63131241762227
element0_dry_mass: [ 369.63131242  369.63131242  369.63131242
    0.            0.            0.            0.
    0.            0.            0.            0.
    0.        ]
element0_eet: 0.0
element0_element_type: Stage
element0_inert_mass: [ 369.63139727  369.63139727  369.63139727
    0.            0.            0.            0.
    0.            0.            0.            0.
    0.        ]
element0_isp_mps: 300.0
element0_isp_rcs: 250.0
element0_loaded_mass: 830.7878473611283
element0_mass_flowrate_mps: 33.990540432597605
element0_mass_flowrate_rcs: 40.78864851911713
element0_max_propellant_mass_mps: 461.156534943506
element0_max_propellant_mass_rcs: 0.0
element0_max_single_burn_prop_mass_mps: 849.1337914791708
element0_max_single_burn_prop_mass_rcs: 10.197162129779283
element0_mga_mass: 0.0
element0_payload_mass: [ 2478.34647561  1478.34647561  1017.18994066
     0.            0.            0.            0.
     0.            0.            0.            0.
     0.        ]
element0_pmr_mass: 0.0
element0_propellant_mass_boiled_mps: 0.0
element0_propellant_mass_boiled_rcs: 0.0
element0_propellant_mass_burned_mps: 461.156534943506
element0_propellant_mass_burned_rcs: 0.0
```

```
element0_propellant_mass_leak_mps: 0.0
element0_propellant_mass_leak_rcs: 0.0
element0_propellant_mass_mps: [ 461.15653494  461.15653494    0.            0.
    0.            0.            0.            0.
    0.            0.            0.            0.        ]
element0_propellant_mass_rcs: [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.]
element0_propellant_mass_reserve_mps: 0.0
element0_propellant_mass_reserve_rcs: 0.0
element0_terminal_event: (10, 0)
element0_thrust_mps: 100000.0
element0_thrust_rcs: 100000.0
element0sub0_inert_mass: 369.63139727421384
element0sub0_mps_isp: 300.0
element0sub0_pmf: 0.6
element0sub0_rcs_isp: 250.0
element1_auto_drop: False
element1_burn_time_mps: 13.5672022002112
element1_burn_time_mps_event1_0: 13.567190791698499
element1_burn_time_rcs: 0.0
element1_burn_time_rcs_event1_0: 0.0
element1_burnout_mass: 369.63131241762227
element1_dry_mass: [ 369.63131242  369.63131242  369.63131242
    0.            0.            0.            0.
    0.            0.            0.            0.
    0.        ]
element1_eet: 0.0
element1_element_type: Stage
element1_inert_mass: [ 369.63139727  369.63139727  369.63139727
    0.            0.            0.            0.
    0.            0.            0.            0.
    0.        ]
element1_isp_mps: 300.0
element1_isp_rcs: 250.0
element1_loaded_mass: 830.7878473611283
element1_mass_flowrate_mps: 33.990540432597605
element1_mass_flowrate_rcs: 40.78864851911713
element1_max_propellant_mass_mps: 461.156534943506
element1_max_propellant_mass_rcs: 0.0
element1_max_single_burn_prop_mass_mps: 849.1337914791708
element1_max_single_burn_prop_mass_rcs: 10.197162129779283
element1_mga_mass: 0.0
element1_payload_mass: [ 2478.34647561  1478.34647561  1017.18994066
     0.            0.            0.            0.
     0.            0.            0.            0.
     0.        ]
element1_pmr_mass: 0.0
element1_propellant_mass_boiled_mps: 0.0
element1_propellant_mass_boiled_rcs: 0.0
element1_propellant_mass_burned_mps: 461.156534943506
element1_propellant_mass_burned_rcs: 0.0
element1_propellant_mass_leak_mps: 0.0
element1_propellant_mass_leak_rcs: 0.0
element1_propellant_mass_mps: [ 461.15653494  461.15653494    0.            0.
    0.            0.            0.            0.
    0.            0.            0.            0.        ]
element1_propellant_mass_rcs: [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.]
element1_propellant_mass_reserve_mps: 0.0
element1_propellant_mass_reserve_rcs: 0.0
element1_terminal_event: (10, 0)
element1_thrust_mps: 100000.0
element1_thrust_rcs: 100000.0
element1sub0_inert_mass: 369.63139727421384
element1sub0_mps_isp: 300.0
element1sub0_pmf: 0.6
element1sub0_rcs_isp: 250.0
element2_auto_drop: False
```

```
element2_boiloff_rate_mps: [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
element2_boiloff_rate_rcs: [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
element2_burn_time_mps: 469396.69810629345
element2_burn_time_mps_event4_0: 469396.7726970796
element2_burn_time_rcs: 0.0
element2_burn_time_rcs_event4_0: 0.0
element2_burnout_mass: 98.0
element2_dry_mass: [ 98.  98.  98.  98.  98.  98.  98.   0.   0.
   0.   0.   0.]
element2_eet: 0.0
element2_element_type: Stage
element2_inert_mass: [ 98.  98.  98.  98.  98.  98.  98.   0.   0.
   0.   0.   0.]
element2_isp_mps: 298.0
element2_isp_rcs: 1.0
element2_loaded_mass: 258.6212830125146
element2_mass_flowrate_mps: 0.0003421866486503115
element2_mass_flowrate_rcs: 0.10197162129779283
element2_max_propellant_mass_mps: 160.62128301251457
element2_max_propellant_mass_rcs: 0.0
element2_max_single_burn_prop_mass_mps: 337.8842861640941
element2_max_single_burn_prop_mass_rcs: 0.10197162129779283
element2_mga_mass: 0.0
element2_mps_start_penalty: 0.0
element2_payload_mass: [ 3050.51303995  2050.51303995  1128.19997007
   388.93734523   377.80853953   377.80853953
   369.63139727     0.             0.             0.
     0.             0.        ]
element2_pmr_mass: 0.0
element2_propellant_mass_boiled_mps: 0.0
element2_propellant_mass_boiled_rcs: 0.0
element2_propellant_mass_burned_mps: 160.62128301251457
element2_propellant_mass_burned_rcs: 0.0
element2_propellant_mass_leak_mps: 0.0
element2_propellant_mass_leak_rcs: 0.0
element2_propellant_mass_mps: [ 160.62128301  160.62128301  160.62128301
  160.62128301  160.62128301    0.            0.
    0.            0.            0.            0.
    0.        ]
element2_propellant_mass_rcs: [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.]
element2_propellant_mass_reserve_mps: 0.0
element2_propellant_mass_reserve_rcs: 0.0
element2_terminal_event: (10, 0)
element2_thrust_mps: 1.0
element2_thrust_rcs: 1.0
element2sub0_inert_mass: 98.0
element2sub0_mass: 98.0
element2sub0_mps_isp: 298.0
element3_auto_drop: False
element3_burn_time_mps: 16.311799541429384
element3_burn_time_mps_event3_0: 0.0
element3_burn_time_mps_event5_0: 0.0
element3_burn_time_mps_event9_0: 16.31180055203983
element3_burn_time_rcs: 0.4733166863348766
element3_burn_time_rcs_event3_0: 0.27284071287463163
element3_burn_time_rcs_event5_0: 0.20047596651912505
element3_burn_time_rcs_event9_0: 0.0
element3_burnout_mass: 369.63139727421384
element3_dry_mass: [ 369.63139727  369.63139727  369.63139727  369.63139727
  369.63139727  369.63139727  369.63139727
  369.63139727  369.63139727  369.63139727
  369.63139727  369.63139727  369.63139727]
element3_eet: 0.0
element3_element_type: Stage
element3_inert_mass: [ 369.63125456  369.63125456  369.63125456
  369.63125456  369.63125456  369.63125456
  369.63125456  369.63125456  369.63125456
  369.63125456  369.63125456  369.63125456]
element3_inert_mass_delta_event8_0: 0.0
element3_isp_mps: 300.0
element3_isp_rcs: 250.0
element3_loaded_mass: 924.0782791155964
element3_mass_flowrate_mps: 33.990540432597605
element3_mass_flowrate_rcs: 40.78864851911713
element3_max_propellant_mass_mps: 554.4468818413825
element3_max_propellant_mass_rcs: 19.305947957146493
element3_max_single_burn_prop_mass_mps: 1350.7316985998618
element3_max_single_burn_prop_mass_rcs: 18.37085441886237
element3_mga_mass: 0.0
element3_payload_mass: [ 2920.19697773  1920.19697773   997.88390785
   258.62128301   258.62128301    98.            98.
     0.          1000.          1000.          1000.
     0.        ]
element3_pmr_mass: 0.0
element3_propellant_mass_boiled_mps: 0.0
element3_propellant_mass_boiled_rcs: 0.0
element3_propellant_mass_burned_mps: 554.4468818413825
element3_propellant_mass_burned_rcs: 19.305947957146493
element3_propellant_mass_delta_event8_0: 0.0
element3_propellant_mass_leak_mps: 0.0
element3_propellant_mass_leak_rcs: 0.0
element3_propellant_mass_mps: [   0.            0.            0.            0.
    0.            0.            0.            0.
    0.          554.44688184    0.            0.        ]
element3_propellant_mass_rcs: [ 19.30594796  19.30594796  19.30594796
  19.30594796   8.17714225   8.17714225   0.
   0.           0.           0.           0.
   0.        ]
element3_propellant_mass_reserve_mps: 0.0
element3_propellant_mass_reserve_rcs: 0.0
element3_terminal_event: (10, 0)
element3_thrust_mps: 100000.0
element3_thrust_rcs: 100000.0
element3_top_off_event8_0: True
element3sub0_inert_mass: 369.6312545609217
element3sub0_mps_isp: 300.0
element3sub0_pmf: 0.6
element3sub0_rcs_isp: 250.0
element4_auto_drop: False
element4_dry_mass: [ 1000.     0.     0.     0.     0.     0.
     0.     0.  1000.  1000.  1000.     0.]
element4_eet: 0.0
element4_element_type: Payload
element4_inert_mass: [ 1000.     0.     0.     0.     0.     0.
     0.     0.  1000.  1000.  1000.     0.]
element4_loaded_mass: 1000.0
element4_mga_mass: 0.0
element4_payload_mass: [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.]
element4_pmr_mass: 0.0
element4_terminal_event: (10, 0)
element4sub0_inert_mass: 1000.0
element4sub0_mass: 1000.0
equivalent_stage_1_0_burn_time_acs: 0.0
equivalent_stage_1_0_burn_time_main: 13.567190791698499
equivalent_stage_1_0_dv: 1500.0
equivalent_stage_1_0_final_mass: 1386.8212530791195
equivalent_stage_1_0_initial_mass: 2309.133547403107
equivalent_stage_1_0_isp_acs: 250.0
equivalent_stage_1_0_isp_main: 300.0
equivalent_stage_1_0_jettison_mass: 0.0
equivalent_stage_1_0_mass_flowrate_acs: 81.57729703823426
equivalent_stage_1_0_mass_flowrate_main: 67.98108086519521
equivalent_stage_1_0_t2w: 8.832024584500271
equivalent_stage_1_0_thrust_acs: 200000.0
```

```
equivalent_stage_1_0_thrust_main: 199999.99999999997
equivalent_stage_3_0_burn_time_acs: 0.0
equivalent_stage_3_0_burn_time_main: 0.27284071287463163
equivalent_stage_3_0_dv: 42.5
equivalent_stage_3_0_final_mass: 636.4298225377794
equivalent_stage_3_0_initial_mass: 647.5586264769281
equivalent_stage_3_0_isp_acs: 250.0
equivalent_stage_3_0_isp_main: 250.0
equivalent_stage_3_0_jettison_mass: 0.0
equivalent_stage_3_0_mass_flowrate_acs: 40.78864851911713
equivalent_stage_3_0_mass_flowrate_main: 40.78864851911713
equivalent_stage_3_0_t2w: 15.747087156042383
equivalent_stage_3_0_thrust_acs: 100000.0
equivalent_stage_3_0_thrust_main: 100000.0
equivalent_stage_4_0_burn_time_acs: 0.0
equivalent_stage_4_0_burn_time_main: 469396.7726970796
equivalent_stage_4_0_dv: 850.0
equivalent_stage_4_0_final_mass: 475.80853952526473
equivalent_stage_4_0_initial_mass: 636.4298480617504
equivalent_stage_4_0_isp_acs: 1.0
equivalent_stage_4_0_isp_main: 298.0
equivalent_stage_4_0_jettison_mass: 0.0
equivalent_stage_4_0_mass_flowrate_acs: 0.10197162129779283
equivalent_stage_4_0_mass_flowrate_main: 0.0003421866486503115
equivalent_stage_4_0_t2w: 0.00016022444831641352
equivalent_stage_4_0_thrust_acs: 1.0
equivalent_stage_4_0_thrust_main: 1.0
equivalent_stage_5_0_burn_time_acs: 0.0
equivalent_stage_5_0_burn_time_main: 0.20047596651912505
equivalent_stage_5_0_dv: 42.5
equivalent_stage_5_0_final_mass: 467.63139727421384
equivalent_stage_5_0_initial_mass: 475.8085410090927
equivalent_stage_5_0_isp_acs: 250.0
equivalent_stage_5_0_isp_main: 250.0
equivalent_stage_5_0_jettison_mass: 0.0
equivalent_stage_5_0_mass_flowrate_acs: 40.78864851911713
equivalent_stage_5_0_mass_flowrate_main: 40.78864851911713
equivalent_stage_5_0_t2w: 21.43122968779246
equivalent_stage_5_0_thrust_acs: 100000.0
equivalent_stage_5_0_thrust_main: 100000.0
equivalent_stage_9_0_burn_time_acs: 0.0
equivalent_stage_9_0_burn_time_main: 16.31180055203983
equivalent_stage_9_0_dv: 1000.0
equivalent_stage_9_0_final_mass: 1369.631397274214
equivalent_stage_9_0_initial_mass: 1924.0783134667918
equivalent_stage_9_0_isp_acs: 250.0
equivalent_stage_9_0_isp_main: 300.0
equivalent_stage_9_0_jettison_mass: 0.0
equivalent_stage_9_0_mass_flowrate_acs: 40.78864851911713
equivalent_stage_9_0_mass_flowrate_main: 33.990540432597605
equivalent_stage_9_0_t2w: 5.299764598149906
equivalent_stage_9_0_thrust_acs: 100000.0
equivalent_stage_9_0_thrust_main: 99999.99999999999
event0_final_mass: 2309.1343229661315
event0_initial_mass: 3309.1343229661315
event10_final_mass: 369.63139727421384
event10_initial_mass: 1369.631397274214
event1_0_dt: 13.567190791698499
event1_dv: 1500.0
event1_final_mass: 1386.8212530791195
event1_initial_mass: 2309.133547403107
event1_propellant_mass_acs: 0.0
event1_propellant_mass_main: 922.3122943239874
event1_sized_dv: 1500.0
event2_final_mass: 647.558628243875
event2_initial_mass: 1386.8212530791195
event3_0_dt: 0.27284071287463163
event3_dv: 42.5
```

```
event3_final_mass: 636.4298225377794
event3_initial_mass: 647.5586264769281
event3_propellant_mass_acs: 0.0
event3_propellant_mass_main: 11.128803939148705
event3_sized_dv: 42.5
event3_system: rcs
event4_0_dt: 469396.7726970796
event4_dv: 850.0
event4_final_mass: 475.80853952526473
event4_initial_mass: 636.4298480617504
event4_propellant_mass_acs: 0.0
event4_propellant_mass_main: 160.62130853648569
event4_sized_dv: 850.0
event5_0_dt: 0.20047596651912505
event5_dv: 42.5
event5_final_mass: 467.63139727421384
event5_initial_mass: 475.8085410090927
event5_propellant_mass_acs: 0.0
event5_propellant_mass_main: 8.177143734878886
event5_sized_dv: 42.5
event5_system: rcs
event6_final_mass: 369.63139727421384
event6_initial_mass: 467.63139727421384
event7_final_mass: 1369.631397274214
event7_initial_mass: 369.63139727421384
event8_final_mass: 1924.0782791155964
event8_initial_mass: 1369.631397274214
event8_mass_type: prop
event8_top_off: True
event9_0_dt: 16.31180055203983
event9_dv: 1000.0
event9_final_mass: 1369.631397274214
event9_initial_mass: 1924.0783134667918
event9_propellant_mass_acs: 0.0
event9_propellant_mass_main: 554.4469161925778
event9_sized_dv: 1000.0
num_elements: 5
num_events: 11
objective_value: 3.3091343229661314
total_payload: [ 1000.     0.     0.     0.     0.     0.
     0.     0.  1000.  1000.  1000.     0.]
vehicle_gross_mass: [ 3309.13432297  2309.13432297  1386.82125308
   647.55862824   636.42982254   475.80853953
   467.63139727   369.63139727  1369.63139727
  1924.07827912  1369.63139727   369.63139727]
```

# APPENDIX C

# MODEL INPUT TABLES

The following tables present the set of input parameters to the various models developed through this body of work. Table 30 consists of mission inputs for the event models and Table 31 consists of the vehicle inputs for the subsystem models. Each input contains a brief description, the type of data expected as the input, the allowable range of input data, units where applicable, and the default value utilized for the experiments and proof of concept problem. Table 33 through Table 53 provide the mappings of the high-level architecture and technology space parameter options to the design space attribute values.

**Table 30:** Event Model Inputs

| **Event** | **Input** | **Type** | **Range** | **Units** | **Default** | **Description** |
|---|---|---|---|---|---|---|
| Burn | acs_factor | float | [0 : 100] | – | 4.0 | Attitude Control System (ACS) factor as a percentage of the total event dv (including FPR) to be added as an attitude control dv utilizing only the RCS |
| Burn | acs_split | float | [0 : 100] | – | 50.0 | percentage of the ACS maneuver dv to perform before the main burn |
| Burn | dv | float | $[0 : +\infty)$ | m/s | 1.0 | the impulsive change in velocity |
| Burn | fpr | float | [0 : 100] | – | 0.0 | Flight Performance Reserve (FPR) as a percentage of the input dv |
| Burn | system | string | ['MPS','RCS'] | – | MPS | propulsion system for the event |
| Idle | dt | float | $[0 : +\infty)$ | days | 1.0 | time lapsed |
| Mass Delta | dm | float | $(-\infty : +\infty)$ | kg | 0.0 | type of mass to add/subtract from the active elements |
| Mass Delta | mass_type | string | ['inert','propellant'] | – | inert | propulsion system for the event |
| Mass Delta | top_off | bool | TRUE, FALSE | – | FALSE | allow the solver to optimize propellant reloading of the active elements |

**Table 31:** SubElement Model Inputs

| Event | Input | Type | Range | Units | Default | Description |
|---|---|---|---|---|---|---|
| Avionics | accuracy | float | [0 : 1] | – | 1.0 | factor to determine sensor accuracy for scaling mass, where higher values correspond to higher accuracy, resulting in increased mass and power requirements |
| Avionics | actuators | list | see description | – | [0,2,2,2] | type of attitude control device(s) as a list of integers in the range [0 : 2] |
| Avionics | additional_devices | list | see description | – | [[0.1,15.]] | list of additional avionics devices, where each item is a list defining a the mass(kg) and power requirement(W) for the device, in the form [m,P] |
| Avionics | comms_type | int | [0 : 2] | – | 2 | type of communications package |
| Avionics | sensors | list | see description | – | [4,1,1,1, 1,1,1,5, 0,0,0] | type of sensor device(s) as a list of integers in the range [0 : 5] |
| Avionics | wireless_sensors | bool | TRUE, FALSE | – | FALSE | wireless transmission of data for sensors within the vehicle |
| Engines | core_type | string | ['PBR','CERMET'] | – | CERMET | type of nuclear core |
| Engines | engine_thrust_mps | float | $[0 : +\infty)$ | kN | 25.0 | thrust per engine of the main propulsion system |

(continued on next page)

(continued from previous page)

| Event | Input | Type | Range | Units | Default | Description |
|---|---|---|---|---|---|---|
| Engines | engine_thrust_rcs | float | $[0 : +\infty)$ | kN | 0.45 | thrust per engine of the reaction control system |
| Engines | isp_mps | float | $[0 : +\infty)$ | sec | 350.0 | the specific impulse of the main propulsion system engines |
| Engines | isp_rcs | float | $[0 : +\infty)$ | sec | 300.0 | the specific impulse of the reaction control system engines |
| Engines | mixture_ratio_mps | float | $[0 : +\infty)$ | – | 3.5 | mass ratio of the oxidizer to fuel of the main propulsion system engines |
| Engines | mixture_ratio_rcs | float | $[0 : +\infty)$ | – | 3.5 | mass ratio of the oxidizer to fuel of the reaction control system engines |
| Engines | mps_class | string | ['liquid', 'solid', 'nuclear', 'electric', 'massless'] | – | liquid | main propulsion system class |
| Engines | P_chamber | float | $[1 : 10]$ | MPa | 3.5 | chamber pressure for nuclear engines |
| Engines | power_mgmt_specific_mass_mps | float | $[0 : +\infty)$ | kg/kW | 6.0 | mass per unit power of the power management system for electric main propulsion systems |
| Engines | power_mgmt_specific_mass_rcs | float | $[0 : +\infty)$ | kg/kW | 6.0 | mass per unit power of the power management system for electric reaction control systems |

(continued on next page)

(continued from previous page)

| Event | Input | Type | Range | Units | Default | Description |
|---|---|---|---|---|---|---|
| Engines | propellants_mps | string | see description | – | LOX/ LCH4 | the oxidizer and Fuel of the propulsion system, separated by a forward slash (/), monopropellant only specify a fuel with no slash (/), fluids must be defined in the fluids definitions model |
| Engines | propellants_rcs | string | see description | – | N2H4 | the oxidizer and Fuel of the propulsion system, separated by a forward slash (/), monopropellant only specify a fuel with no slash (/), fluids must be defined in the fluids definitions model |
| Engines | rcs_class | string | ['liquid', 'electric', 'massless'] | – | liquid | reaction control system class |
| Engines | redundancy_mps | float | $[0 : +\infty)$ | – | 0.2 | redundancy of electric thrusters in the main propulsion system, where a value of 1 corresponds to dual redundancy |
| Engines | redundancy_rcs | float | $[0 : +\infty)$ | – | 0.2 | redundancy of electric thrusters in the reaction control system, where a value of 1 corresponds to dual redundancy |

(continued on next page)

(continued from previous page)

| **Event** | **Input** | **Type** | **Range** | **Units** | **Default** | **Description** |
|---|---|---|---|---|---|---|
| Engines | start_penalty_mps | float | $[0 : +\infty)$ | kg | 40.0 | mass of propellant lost during engine startup of the main propulsion system |
| Engines | start_penalty_rcs | float | $[0 : +\infty)$ | kg | 1.0 | mass of propellant lost during engine startup of the reaction control system |
| Engines | T_chamber | float | varies | K | 2800.0 | chamber temperature of the nuclear engines, range varies based on propellant type |
| Engines | thruster_efficiency_mps | float | $[0 : 1]$ | – | 0.5 | electric thruster efficiency of the main propulsion system |
| Engines | thruster_efficiency_rcs | float | $[0 : 1]$ | – | 0.5 | electric thruster efficiency of the reaction control system |
| Engines | thruster_power_mps | float | $[0 : +\infty)$ | kW | 1.0 | input power required per thruster of the main propulsion system |
| Engines | thruster_power_rcs | float | $[0 : +\infty)$ | kW | 1.0 | input power required per thruster of the reaction control system |
| Engines | thruster_specific_mass_mps | float | $[0 : +\infty)$ | kg/kW | 7.0 | electric thruster mass per unit power of the main propulsion system |
| Engines | thruster_specific_mass_rcs | float | $[0 : +\infty)$ | kg/kW | 7.0 | electric thruster mass per unit power of the reaction control system |

(continued on next page)

(continued from previous page)

| Event | Input | Type | Range | Units | Default | Description |
|---|---|---|---|---|---|---|
| Engines | total_thrust_mps | float | $[0 : +\infty)$ | kN | 100.0 | total thrust of the main propulsion system |
| Engines | total_thrust_rcs | float | $[0 : +\infty)$ | kN | 1.8 | total thrust of the reaction control system |
| Power | array_density | float | $[0 : +\infty)$ | $kg/m^2$ | 3.5 | aerial density of the solar array, including PV cells and array structure |
| Power | cell_degradation | float | [0 : 100] | %/yea | 3.75 | decrease in cell power production |
| Power | cell_efficiency | float | [0 : 1] | – | 0.175 | efficiency of the PV cell at converting solar energy to electrical energy |
| Power | discharge_depth | float | [0 : 1] | – | 0.2 | depth of discharge of the battery storage system as a fraction of the storage capacity |
| Power | energy_tracking | string | ['direct', 'peak-tracking'] | – | peak-tracking | energy tracking scheme for the solar array system |
| Power | generator_type | string | ['pv', 'rtg', 'none'] | – | pv | type of power generation |
| Power | low_array_degradation | bool | TRUE, FALSE | – | FALSE | reduces solar array degradation due to assembly, configuration (shadowing), and operational temperature |

(continued on next page)

(continued from previous page)

| Event | Input | Type | Range | Units | Default | Description |
|---|---|---|---|---|---|---|
| Power | max_eclipse | float | varies | hours | 0.5 | maximum eclipse time during the mission for which batteries must provide power, must be less than orbit_period |
| Power | mission_duration | float | $[0 : +\infty)$ | years | 0.5 | the duration of the mission |
| Power | ops_distance | float | $[0 : +\infty)$ | AU | 1.0 | solar distance from the sun of the worst case operational environment |
| Power | orbit_period | float | $[0 : +\infty)$ | hours | 2.0 | the orbit period relating to the eclipse period |
| Power | storage_specific_energy | float | $[0 : +\infty)$ | Wh/kg | 30.0 | energy density of the power storage system |
| Power | transmission_efficiency | float | $[0 : 1)$ | – | 0.9 | efficiency of transmitting power from the generator/power storage to the load |
| Structures | A_de | float | $[0 : +\infty)$ | $m^2$ | -1.0 | surface area of design-envelop volume of the element |
| Structures | adapter | bool | TRUE, FALSE | – | FALSE | sizes an adapter instead of a primary/secondary structure, overriding the 'manned' or 'truss' input settings |

(continued on next page)

(continued from previous page)

| **Event** | **Input** | **Type** | **Range** | **Units** | **Default** | **Description** |
|---|---|---|---|---|---|---|
| Structures | composite | bool | TRUE, FALSE | – | FALSE | reduces overall mass by 30% for composite structures |
| Structures | manned | bool | TRUE, FALSE | – | FALSE | size structures for man-rated vehicles |
| Structures | truss | bool | TRUE, FALSE | – | FALSE | assume a truss structure for a single tank, otherwise, assume an in-line tank structure |
| Tanks | composite_fuel_tanks_mps | bool | TRUE, FALSE | – | FALSE | utilizes composite materials for the MPS fuel tanks |
| Tanks | composite_fuel_tanks_rcs | bool | TRUE, FALSE | – | FALSE | utilizes composite materials for the RCS fuel tanks |
| Tanks | composite_ox_tanks_mps | bool | TRUE, FALSE | – | FALSE | utilizes composite materials for the MPS oxidizer tanks |
| Tanks | composite_ox_tanks_rcs | bool | TRUE, FALSE | – | FALSE | utilizes composite materials for the RCS oxidizer tanks |
| Tanks | copv_pressurant_tanks | bool | TRUE, FALSE | – | FALSE | utilizes composite overwrap pressure vessels for the pressurant tanks |
| Tanks | fuel_pressure_mps | float | $(0 : +\infty)$ | MPa | 40.0 | main propulsion system fuel tank pressures |
| Tanks | fuel_pressure_rcs | float | $(0 : +\infty)$ | MPa | 40.0 | reaction control system fuel tank pressures |

(continued on next page)

(continued from previous page)

| Event | Input | Type | Range | Units | Default | Description |
|---|---|---|---|---|---|---|
| Tanks | ivfm | bool | TRUE, FALSE | – | FALSE | integrated vehicle fluid management |
| Tanks | ld_ratio_fuel_tanks_mps | float | $(0 : +\infty)$ | – | 1.0 | ratio of the length over the diameter of the main propulsion system fuel tanks |
| Tanks | ld_ratio_fuel_tanks_rcs | float | $(0 : +\infty)$ | – | 1.0 | ratio of the length over the diameter of the reaction control system fuel tanks |
| Tanks | ld_ratio_ox_tanks_mps | float | $(0 : +\infty)$ | – | 1.0 | ratio of the length over the diameter of the main propulsion system oxidizer tanks |
| Tanks | ld_ratio_ox_tanks_rcs | float | $(0 : +\infty)$ | – | 1.0 | ratio of the length over the diameter of the reaction control system oxidizer tanks |
| Tanks | material_density | float | $(0 : +\infty)$ | $kg/m^3$ | 2685.0 | density of the tank material |
| Tanks | material_strength | float | $(0 : +\infty)$ | MPa | 538.0 | ultimate strength of the tank material |
| Tanks | num_fuel_tanks_mps | int | $[0 : +\infty)$ | – | 2 | number of fuel tanks for the main propulsion system |
| Tanks | num_fuel_tanks_rcs | int | $[0 : +\infty)$ | – | 1 | number of fuel tanks for the reaction control system |

(continued on next page)

(continued from previous page)

| Event | Input | Type | Range | Units | Default | Description |
|---|---|---|---|---|---|---|
| Tanks | num_ox_tanks_mps | int | $[0:+\infty)$ | – | 2 | number of oxidizer tanks for the main propulsion system |
| Tanks | num_ox_tanks_rcs | int | $[0:+\infty)$ | – | 1 | number of oxidizer tanks for the reaction control system |
| Tanks | num_tanks_pressurant | int | $[0:+\infty)$ | – | 2 | number of tanks for the pressurant |
| Tanks | ox_pressure_mps | float | $(0:+\infty)$ | MPa | 50.0 | main propulsion system oxidizer tank pressures |
| Tanks | ox_pressure_rcs | float | $(0:+\infty)$ | MPa | 50.0 | reaction control system oxidizer tank pressures |
| Tanks | pressurant | string | see description | – | H2 | pressurant fluid for the propulsion systems, must be defined in the fluid definitions |
| Tanks | pressurant_pressure | float | $(0:+\infty)$ | MPa | 41.3 | initial pressure of the pressurant tanks |
| Tanks | separator_type_mps | string | ['pmd', 'ped'] | – | pmd | type of device used to separate pressurant gas and liquid propellant in the main propulsion system propellant tanks |

(continued on next page)

(continued from previous page)

| Event | Input | Type | Range | Units | Default | Description |
|---|---|---|---|---|---|---|
| Tanks | separator_type_rcs | string | ['pmd', 'ped'] | – | pmd | type of device used to separate pressurant gas and liquid propellant in the reaction control system propellant tanks |
| Tanks | tank_ld_ratio_pressurant | float | $(0 : +\infty)$ | – | 1.0 | ratio of the length over the diameter of the pressurant tanks |
| Thermal | active_cooling_mps | bool | TRUE, FALSE | – | FALSE | include an active cooling system to reduce propellant boil off to zero for the main propulsion system |
| Thermal | active_cooling_rcs | bool | TRUE, FALSE | – | FALSE | include an active cooling system to reduce propellant boil off to zero for the reaction control system |
| Thermal | albedo | float | $[0 : 1]$ | – | 0.12 | bond albedo of the orbited body, If not orbiting a body, set this value to zero and any value for orbit_alt and orbit_radius |
| Thermal | deep_space | bool | TRUE, FALSE | – | FALSE | ignore radiation affect near an orbited body |

(continued on next page)

(continued from previous page)

| Event | Input | Type | Range | Units | Default | Description |
|---|---|---|---|---|---|---|
| Thermal | external_tanks | bool | TRUE, FALSE | – | FALSE | propellant tanks are assumed external to the main element structure and are directly affected by external radiation sources such as the Sun and/or orbited bodies. MLI mass and energy leak will be calculated based on tank geometry. If False, MLI mass and energy leak are calculated based on assumed element geometry |
| Thermal | hi_efficiency_radiators | bool | TRUE, FALSE | – | FALSE | uses hi-efficiency radiators with high emissivity and high fin efficiency to radiate heat at a greater rate for an equivalent radiator area |
| Thermal | mli_layers_mps | int | $[10 : +\infty)$ | – | 60 | number of layers in the MLI blankets for the main propulsion system tanks |
| Thermal | mli_layers_rcs | int | $[10 : +\infty)$ | – | 60 | number of layers in the MLI blankets for the reaction control system tanks |
| Thermal | ops_distance | float | $[0 : +\infty)$ | AU | 1.0 | solar distance from the sun of the worst case operational environment |

(continued on next page)

(continued from previous page)

| **Event** | **Input** | **Type** | **Range** | **Units** | **Default** | **Description** |
|---|---|---|---|---|---|---|
| Thermal | orbit_alt | float | $[0:+\infty)$ | km | 500.0 | orbit altitude of the element above the orbited body in the worst thermal environment |
| Thermal | r_body | float | $(0:+\infty)$ | km | 1737.0 | radius of the orbited body |
| Thermal | radiator_density | float | $[0:+\infty)$ | $kg/m^2$ | 4.5 | aerial density of the thermal radiators |
| Thermal | T_body | float | $(0:+\infty)$ | K | 270.0 | average temperature of the orbited body |

**Table 32:** Cost Model Inputs

| **Input** | **Type** | **Range** | **Units** | **Description** |
|---|---|---|---|---|
| num_stages | int | [1 : 3] | – | number of vehicle stages |
| class | str | – | – | the class of each stage |
| propellant | str | – | – | propellant type of the main propulsion system of each stage |
| prop_mass | float | $(0 : +\infty)$ | kg | mass of propellant for each stage |
| num_engines | int | $[0 : +\infty)$ | – | number of engines in the main propulsion system of each stage |
| engines_mass | float | $[0 : +\infty)$ | kg | total mass of the engines subsystem |
| burnout_mass | float | $[0 : +\infty)$ | kg | total mass of each stage less burned propellant |
| structure_type | str | – | – | type of primary structure of each stage, either manned or unmanned |
| engine_tech | bool | TRUE, FALSE | – | logical control for applied engine technologies |
| veh_tech | bool | TRUE, FALSE | – | logical control for other applied vehicle technologies |

**Table 33:** Number of Stages Option To Design Attribute Value Mappings for Experiments 1-4

| | **Attribute Values** | |
|---|---|---|
| **Parameter Option** | event_list | element_list |
| 1 | default minus events 4 and 7 | [s0,payload] |
| 2 | default minus event 7 | [s0,s1,payload] |
| 3 | default | [s0,s1,payload] |

**Table 34:** Number of Stages Option To Design Attribute Value Mappings for Proof of Concept

| | **Attribute Values** | |
|---|---|---|
| **Parameter Option** | event_list | element_list |
| 1 | default minus events 4, 11, 23, 24, 25, 26, 27, and 28 | [s0,payload] |
| 2 | default minus events 4, 11, 23, 24, 25, 26, 27, and 28 if MPS Class is nuclear otherwise default minus events 4, 11, 26, 27, and 28 | [s0,s1,payload] |
| 3 | default minus events 11, 23, 24, 25, 26, 27, and 28 if MPS Class is nuclear otherwise default | [s0,s1,s3,payload] |

**Table 35:** Payload Mass Option To Design Attribute Values Mappings

| | **Attribute Value** |
|---|---|
| **Parameter Option** | mass (kg) |
| 1000 | 1000.0 |
| 10000 | 10000.0 |

**Table 36:** MPS Class Option To Design Attribute Value Mappings

| | **Attribute Values** | | |
|---|---|---|---|
| **Parameter Option** | start_penalty_mps (kg) | total_thrust_mps (kN) | engine_thrust_mps (kN) |
| liquid | 40.0 | 100.0 | 25.0 |
| solid | 0.0 | 100.0 | 100.0 |
| nuclear | 40.0 | 300.0 | 100.0 |
| electric | 1.0 | 0.1 | – |

**Table 37:** MPS Propellant Option To Design Attribute Value Mappings

| Parameter Option | Attribute Values<br>isp_mps (s) | mixture_ratio_mps |
|---|---|---|
| $LOX/LH_2$ | 425.0 | 6.0 |
| $LOX/LCH_4$ | 350.0 | 3.5 |
| NTO/MMH | 300.0 | 2.16 |
| Xenon | 1000.0 | 1.0 |
| $LH_2$ | 900.0 | 1.0 |
| $N_2/H_4$ | 275.0 | 1.0 |

**Table 38:** RCS Class Option To Design Attribute Value Mappings

| Parameter Option | Attribute Values<br>start_penalty_rcs (kg) | total_thrust_rcs (kN) | engine_thrust_rcs (kN) |
|---|---|---|---|
| liquid | 1.0 | 1.8 | 0.45 |
| electric | 0.1 | 0.001 | – |

**Table 39:** RCS Propellant Option To Design Attribute Value Mappings

| Parameter Option | Attribute Values<br>isp_mps (s) | mixture_ratio_mps |
|---|---|---|
| $LOX/LH_2$ | 400.0 | 6.0 |
| $LOX/LCH_4$ | 300.0 | 3.5 |
| NTO/MMH | 290.0 | 2.16 |
| Xenon | 2000.0 | 1.0 |
| $N_2/H_4$ | 275.0 | 1.0 |

**Table 40:** Pressurant Option To Design Attribute Value Mappings

| Parameter Option | Attribute Values<br>pressurant |
|---|---|
| Helium | helium |
| Nitrogen | nitrogen |
| None | none |

**Table 41:** Tank Configuration Option To Design Attribute Value Mappings

| | **Attribute Values** | |
|---|---|---|
| **Parameter Option** | num_fuel_tanks_mps | num_ox_tanks_mps |
| Stacked | 1 | 1 |
| Disk | 2 | 2 |
| Single | 1 | 0 |

**Table 42:** Structure Type Option To Design Attribute Value Mappings

| | **Attribute Values** |
|---|---|
| **Parameter Option** | manned |
| Manned | TRUE |
| Unmanned | FALSE |

**Table 43:** Power System Option To Design Attribute Value Mappings

| | **Attribute Values** |
|---|---|
| **Parameter Option** | generator_type |
| Solar | solar |
| RTG | rtg |

**Table 44:** MLI Layers Option To Design Attribute Value Mappings

| | **Attribute Values** | |
|---|---|---|
| **Parameter Option** | mli_layers_mps | mli_layers_rcs |
| 10 | 10 | 10 |
| 20 | 20 | 20 |
| 30 | 30 | 30 |
| 50 | 50 | 50 |
| 60 | 60 | 60 |

**Table 45:** Communications Type Option To Design Attribute Value Mappings

| | **Attribute Values** |
|---|---|
| **Parameter Option** | comms_type |
| None | 0 |
| Near Earth | 1 |
| Deep Space | 2 |

**Table 46:** Wireless Sensors Option To Design Attribute Value Mappings

| | **Attribute Values** |
|---|---|
| **Parameter Option** | wireless_sensors |
| TRUE | TRUE |
| FALSE | FALSE |

**Table 47:** Low Leak Valves Option To Design Attribute Value Mappings

| | **Attribute Values** | |
|---|---|---|
| **Parameter Option** | start_penalty_mps (kg) | start_penalty_rcs (kg) |
| TRUE | 0.0 | 0.0 |
| FALSE | no change | no change |

**Table 48:** High Capacity Energy Storage Option To Design Attribute Value Mappings

| | **Attribute Values** |
|---|---|
| **Parameter Option** | storage_specific_energy (kWh/kg) |
| TRUE | 125.0 |
| FALSE | 30.0 |

**Table 49:** Composite Structures Option To Design Attribute Value Mappings

| | **Attribute Values** |
|---|---|
| **Parameter Option** | composite |
| TRUE | TRUE |
| FALSE | FALSE |

**Table 50:** Composite Propellant Tanks Option To Design Attribute Value Mappings

| | **Attribute Values** | | | |
|---|---|---|---|---|
| **Parameter Option** | composite_fuel _tanks_mps | composite_ox _tanks_mps | composite_fuel _tanks_rcs | composite_ox _tanks_rcs |
| TRUE | TRUE | TRUE | TRUE | TRUE |
| FALSE | FALSE | FALSE | FALSE | FALSE |

**Table 51:** Integrated MPS/RCS Propellant Storage Option To Design Attribute Value Mappings

| | **Attribute Values** |
|---|---|
| **Parameter Option** | ivfm |
| TRUE | TRUE |
| FALSE | FALSE |

**Table 52:** Autogenous Pressurization Option To Design Attribute Value Mappings

| | **Attribute Values** |
|---|---|
| **Parameter Option** | pressurant |
| TRUE | none |
| FALSE | no change |

**Table 53:** Active Cryocooling Option To Design Attribute Value Mappings

| | **Attribute Values** | |
|---|---|---|
| **Parameter Option** | active_cooling_mps | active_cooling_rcs |
| TRUE | TRUE | TRUE |
| FALSE | FALSE | FALSE |

# APPENDIX D

# DEFAULT DYREQT MODEL INPUTS

The following are default inputs provided to DYREQT for the experiments and the proof of concept performed for this dissertation. The experiments utilized a notional round-trip Mars mission performed by a vehicle with up to three unique stages. The proof of concept utilized a notional manned 2033 Mars fly-by mission with up to a three-stage vehicle. For both the experiments and the proof of concept, these default inputs are altered based on the specific architecture and technology space options to represent a variety of mission/vehicle combinations to be evaluated. Alternatives are limited to a maximum of three stages in both the experiments and the proof of concept, but may contain fewer stages. The values represented in these default inputs are those used for design space attributes which are not mapped to any architecture space parameters.

## *D.1 Experimentation Default Mission Inputs*

```
####################
# MISSION DEF
####################
# default mission/conops description
DESC = 'Roundtrip'

# default mission input
TDI = [{'event_type':'Burn','params':{'dv':{'val':1000.,'units':'m/s'},
        'system':'MPS','acs_factor':4.0}}, #TDI burn
       {'event_type':'Idle','params':{'dt':{'val':20.,'units':'d'}}}, # transit
       {'event_type':'Burn','params':{'dv':{'val':50.,'units':'m/s'},
        'system':'RCS'}}, # course correction
       {'event_type':'Idle','params':{'dt':{'val':20.,'units':'d'}}}, # transit
       {'event_type':'Drop','params':{}}] # drop TDI stage
DOI = [{'event_type':'Burn','params':{'dv':{'val':1500.,'units':'m/s'},
        'system':'MPS','acs_factor':4.0}}, # DOI burn
       {'event_type':'Idle','params':{'dt':{'val':50.,'units':'d'}}},
       {'event_type':'Drop','params':{}}] # drop DOI stage
TRI = [{'event_type':'Burn','params':{'dv':{'val':800.,'units':'m/s'},
        'system':'MPS','acs_factor':4.0}}, #TRI burn
       {'event_type':'Idle','params':{'dt':{'val':20.,'units':'d'}}}, # transit
       {'event_type':'Burn','params':{'dv':{'val':50.,'units':'m/s'},
        'system':'RCS'}}, # course correction
       {'event_type':'Idle','params':{'dt':{'val':20.,'units':'d'}}}] # transit
```

```
ROI = [{'event_type':'Burn','params':{'dv':{'val':600.,'units':'m/s'},
        'system':'MPS','acs_factor':4.0}},
       {'event_type':'Drop','params':{}}] # drop TRI/ROI stage

event_list = TDI + DOI + TRI + ROI
MISSION = {'event_list':event_list}

# default conops input
TDI = [[{'active_elements':[0]}],
       [{'active_elements':[]}],
       [{'active_elements':[0]}],
       [{'active_elements':[]}],
       [{'active_elements':[0]}]]
DOI = [[{'active_elements':[1]}],
       [{'active_elements':[]}],
       [{'active_elements':[1]}]]
TRI = [[{'active_elements':[2]}],
       [{'active_elements':[]}],
       [{'active_elements':[2]}],
       [{'active_elements':[]}]]
ROI = [[{'active_elements':[2]}],
       [{'active_elements':[2]}]]

CONOPS = TDI + DOI + TRI + ROI
```

## D.2 Experimentation Default Vehicle Inputs

```
###################
# VEHICLE DEF
###################
# default mission/conops description
DESC = 'Default'

# vehicle input
avionics = {'subelement_type':'AvionicsPhD',
            'params':{'actuators':[0,2,2,2], # 1 reaction wheel and 3 mag torquers
                      'sensors':[4,1,1,1,1,1,1,5,0,0,0], # 1 horizon sensor, 1 magnetometer, 3 gyros,
                      ↪  6 sun sensors
                      'comms_type':2, # deep space
                      'accuracy':1., # highest accuracy
                      'wireless_sensors':False,
                      'additional_devices':[[0.1,15.]] # main cpu
                     }
           }
engines = {'subelement_type':'EnginesPhD',
           'params':{'mps_class':'liquid',
                     'rcs_class':'liquid',
                     'propellants_mps':'lox/lch4',
                     'propellants_rcs':'hydrazine',
                     'start_penalty_mps':{'val':40.,'units':'kg'},
                     'start_penalty_rcs':{'val':1.,'units':'kg'},
                     'total_thrust_mps':{'val':100.,'units':'kN'},
                     'total_thrust_rcs':{'val':1.8,'units':'kN'},
                     'isp_mps':{'val':350.,'units':'s'},
                     'isp_rcs':{'val':300.,'units':'s'},
                     'engine_thrust_mps':{'val':25.,'units':'kN'},
                     'engine_thrust_rcs':{'val':0.45,'units':'kN'},
                     'mixture_ratio_mps':3.5,
                     'mixture_ratio_rcs':3.5,
                     'core_type':'cermet',
                     'T_chamber':{'val':2800.,'units':'K'},
                     'P_chamber':{'val':3.5,'units':'MPa'},
                     'thruster_efficiency_mps':0.5,
                     'thruster_efficiency_rcs':0.5,
                     'thruster_specific_mass_mps':{'val':7.,'units':'kg/kW'},
                     'thruster_specific_mass_rcs':{'val':7.,'units':'kg/kW'},
                     'thruster_power_mps':{'val':1.,'units':'kW'},
                     'thruster_power_rcs':{'val':1.,'units':'kW'},
                     'redundancy_mps':0.2,
                     'redundancy_rcs':0.2,
                     'power_mgmt_specific_mass_mps':{'val':6.,'units':'kg/kW'},
                     'power_mgmt_specific_mass_rcs':{'val':6.,'units':'kg/kW'},
                    }
          }
power = {'subelement_type':'PowerPhD',
         'params':{'generator_type':'PV',
                   'transmission_efficiency':0.9,
                   'cell_efficiency':0.175,
                   'cell_degradation':{'val':3.75,'units':'1/yr'},
                   'array_density':{'val':3.5, 'units':'kg/m**2'},
                   'discharge_depth':0.2,
                   'storage_specific_energy':{'val':30.,'units':'W*h/kg'},
                   'energy_transfer':'peak-tracking',
                   'low_array_degradation':False,
                   'orbit_period':{'val':2.,'units':'h'}, # low lunar orbit
                   'max_eclipse':{'val':0.5,'units':'h'}, # low lunar orbit
                   'ops_distance':{'val':1.,'units':'AU'}, # low lunar orbit
                   'mission_duration':{'val':0.5,'units':'yr'}
                  }
        }
structures = {'subelement_type':'StructuresPhD',
              'params':{'manned':False,
                        'truss':False,
                        'adapter':False,
                        'composite':False,
                        'A_de':-1.
                        }
            }
tanks = {'subelement_type':'TanksPhD',
         'params':{'num_fuel_tanks_mps':2,
                   'num_ox_tanks_mps':2,
                   'fuel_pressure_mps':{'val':40.,'units':'psi'},
                   'ox_pressure_mps':{'val':50.,'units':'psi'},
                   'ld_ratio_fuel_tanks_mps':1.,
                   'ld_ratio_ox_tanks_mps':1.,
                   'separator_type_mps':'pmd',
                   'num_fuel_tanks_rcs':1,
                   'num_ox_tanks_rcs':1,
                   'fuel_pressure_rcs':{'val':40.,'units':'psi'},
                   'ox_pressure_rcs':{'val':50.,'units':'psi'},
                   'ld_ratio_fuel_tanks_rcs':1.,
                   'ld_ratio_ox_tanks_rcs':1.,
                   'separator_type_rcs':'pmd',
                   'pressurant':'Helium',
                   'pressurant_pressure':{'val':6000.,'units':'psi'},
                   'num_tanks_pressurant':2,
                   'tank_ld_ratio_pressurant':1.0,
                   'material_strength':{'val':538.,'units':'MPa'},
                   'material_density':{'val':2685.,'units':'kg/m**3'},
                   'copv_pressurant_tank':False,
                   'composite_fuel_tanks_mps':False,
                   'composite_ox_tanks_mps':False,
                   'composite_fuel_tanks_rcs':False,
                   'composite_ox_tanks_rcs':False,
                   'ivfm':False,
                  }
        }
thermal = {'subelement_type':'ThermalPhD',
           'params':{'mli_layers_mps':60,
                     'mli_layers_rcs':60,
                     'active_cooling_mps':False,
                     'active_cooling_rcs':False,
                     'radiator_density':{'val':4.5,'units':'kg/m**2'},
                     'external_tanks':False,
                     'hi_efficiency_radiators':False,
                     'ops_distance':{'val':1.0,'units':'AU'},
                     'deep_space':False,
                     'orbit_alt':{'val':500.,'units':'km'}, # low lunar
                     'r_body':{'val':1737.,'units':'km'}, # moon
                     'T_body':{'val':270.,'units':'K'}, # moon
                     'albedo':0.12 # moon
                    }
          }
# fixed mass subsystem for payload
submass = {'subelement_type':'FixedMass',
           'params':{'mass':{'val':1000.0,'units':'kg'}}}
stage = {'element_type':'Stage',
         'subelement_list':[avionics,engines,power,structures,tanks,thermal],
         'params':{'auto_drop':False,'mps_reserve':3.,'rcs_reserve':3.,
                   'boiloff_model':'constant-rate','mga':20.0}}
s0 = deepcopy(stage)
s1 = deepcopy(stage)
s2 = deepcopy(stage)
payload = {'element_type':'Payload','subelement_list':[submass],}
element_list = [s0,s1,s2,payload]
VEHICLE = {'element_list':element_list}
```

## D.3 Proof of Concept Default Mission Inputs

```
###################
# MISSION DEF
###################

# mission input, 2033 flight
E00 = {'event_type':'Idle','params':{'dt':{'val':145.,'units':'d'}}}
E01 = {'event_type':'Burn','params':{'dv':{'val':220.,'units':'m/s'},'system':'RCS'}} # TCMDRO_33
E02 = {'event_type':'Idle','params':{'dt':{'val':200.,'units':'d'}}}
E03 = {'event_type':'Burn','params':{'dv':{'val':629.,'units':'m/s'},'system':'MPS'}} # TMI_33_1
E04 = {'event_type':'Drop','params':{}} # drop TMI_33
E05 = {'event_type':'Idle','params':{'dt':{'val':130.,'units':'d'}}}
E06 = {'event_type':'Burn','params':{'dv':{'val':40.,'units':'m/s'},'system':'RCS'}} # TCM1_33
E07 = {'event_type':'Idle','params':{'dt':{'val':131.,'units':'d'}}}
E08 = {'event_type':'Drop','params':{}} # drop ConMOI_33
E09 = {'event_type':'Idle','params':{'dt':{'val':1.,'units':'d'}}}
E10 = {'event_type':'Burn','params':{'dv':{'val':1290.,'units':'m/s'},'system':'MPS'}} # Flyby_33
E11 = {'event_type':'Drop','params':{}} # drop FlyBy_33
E12 = {'event_type':'Idle','params':{'dt':{'val':159.,'units':'d'}}}
E13 = {'event_type':'Burn','params':{'dv':{'val':40.,'units':'m/s'},'system':'RCS'}} # TCM2_33
E14 = {'event_type':'Idle','params':{'dt':{'val':159.,'units':'d'}}}
E15 = {'event_type':'Drop','params':{}} # drop ConEOI_33
E16 = {'event_type':'Burn','params':{'dv':{'val':1072.,'units':'m/s'},'system':'MPS'}} # EOI_33
E17 = {'event_type':'Drop','params':{}} # drop ConRd_33
E18 = {'event_type':'Burn','params':{'dv':{'val':220.,'units':'m/s'},'system':'RCS'}} # EOI_33
E19 = {'event_type':'Idle','params':{'dt':{'val':200.,'units':'d'}}}
E20 = {'event_type':'Drop','params':{}} # drop DSH
E21 = {'event_type':'Burn','params':{'dv':{'val':5.,'units':'m/s'},'system':'RCS'}} # EOI_33 Disposal
E22 = {'event_type':'Drop','params':{}} # drop EOI_33
E23 = {'event_type':'Connect','params':{}} # connect Flyby_33
E24 = {'event_type':'Burn','params':{'dv':{'val':5.,'units':'m/s'},'system':'RCS'}} # Flyby_33
↪  Disposal
E25 = {'event_type':'Drop','params':{}} # drop Flyby_33_33
E26 = {'event_type':'Connect','params':{}} # connect TMI_33
E27 = {'event_type':'Burn','params':{'dv':{'val':5.,'units':'m/s'},'system':'RCS'}} # TMI_33 Disposal
E28 = {'event_type':'Drop','params':{}} # drop TMI_33_33
E29 = {'event_type':'Connect','params':{}} # connect DSH
# 2033 mission
event_list = [E00,E01,E02,E03,E04,E05,E06,E07,E08,E09,E10,E11,E12,E13,E14,
              E15,E16,E17,E18,E19,E20,E21,E22,E23,E24,E25,E26,E27,E28,E29]
MISSION = {'event_list':event_list}

# 2033 conops
CONOPS = [[{'active_elements':[]}],                          # E00
          [{'active_elements':[0]}],                         # E01
          [{'active_elements':[]}],                          # E02
          [{'active_elements':[0]}],                         # E03
          [{'active_elements':[0]}],                         # E04
          [{'active_elements':[]}],                          # E05
          [{'active_elements':[1]}],                         # E06
          [{'active_elements':[]}],                          # E07
          [{'active_elements':[5]}],                         # E08
          [{'active_elements':[]}],                          # E09
          [{'active_elements':[1]}],                         # E10
          [{'active_elements':[1]}],                         # E11
          [{'active_elements':[]}],                          # E12
          [{'active_elements':[2]}],                         # E13
          [{'active_elements':[]}],                          # E14
          [{'active_elements':[6]}],                         # E15
          [{'active_elements':[2]}],                         # E16
          [{'active_elements':[4]}],                         # E17
          [{'active_elements':[2]}],                         # E18
          [{'active_elements':[]}],                          # E19
          [{'active_elements':[3]}],                         # E20
          [{'active_elements':[2]}],                         # E21
          [{'active_elements':[2]}],                         # E22
          [{'active_elements':[1]}],                         # E23
```

```
            [{'active_elements':[1]}],                         # E24
            [{'active_elements':[1]}],                         # E25
            [{'active_elements':[0]}],                         # E26
            [{'active_elements':[0]}],                         # E27
            [{'active_elements':[0]}],                         # E28
            [{'active_elements':[3]}],                         # E29
           ]

# default vehicle
# index          | 0| 1| 2| 3 |   4    |    5    |    6    |
# element_list = [s0,s1,s2,DSH,ConRd_33,ConMOI_33,ConEOI_33]
```

## D.4 Proof of Concept Default Vehicle Inputs

```
###################
# VEHICLE DEF
###################

# vehicle input
avionics = {'subelement_type':'AvionicsPhD',
            'params':{'actuators':[0,2,2,2], # 1 reaction wheel and 3 mag torquers
                      'sensors':[4,1,1,1,1,1,1,5,0,0,0], # 1 horizon sensor, 1 magnetometer, 3 gyros,
                      ↪  6 sun sensors
                      'comms_type':2, # deep space
                      'accuracy':1., # highest accuracy
                      'wireless_sensors':False,
                      'additional_devices':[[0.1,15.]] # main cpu
                     }
           }
engines = {'subelement_type':'EnginesPhD',
           'params':{'mps_class':'liquid',
                     'rcs_class':'liquid',
                     'propellants_mps':'lox/lch4',
                     'propellants_rcs':'hydrazine',
                     'start_penalty_mps':{'val':40.,'units':'kg'},
                     'start_penalty_rcs':{'val':1.,'units':'kg'},
                     'total_thrust_mps':{'val':100.,'units':'kN'},
                     'total_thrust_rcs':{'val':1.8,'units':'kN'},
                     'isp_mps':{'val':350.,'units':'s'},
                     'isp_rcs':{'val':300.,'units':'s'},
                     'engine_thrust_mps':{'val':25.,'units':'kN'},
                     'engine_thrust_rcs':{'val':0.45,'units':'kN'},
                     'mixture_ratio_mps':3.5,
                     'mixture_ratio_rcs':3.5,
                     'core_type':'cermet',
                     'T_chamber':{'val':2800.,'units':'K'},
                     'P_chamber':{'val':3.5,'units':'MPa'},
                     'thruster_efficiency_mps':0.5,
                     'thruster_efficiency_rcs':0.5,
                     'thruster_specific_mass_mps':{'val':7.,'units':'kg/kW'},
                     'thruster_specific_mass_rcs':{'val':7.,'units':'kg/kW'},
                     'thruster_power_mps':{'val':1.,'units':'kW'},
                     'thruster_power_rcs':{'val':1.,'units':'kW'},
                     'redundancy_mps':0.2,
                     'redundancy_rcs':0.2,
                     'power_mgmt_specific_mass_mps':{'val':6.,'units':'kg/kW'},
                     'power_mgmt_specific_mass_rcs':{'val':6.,'units':'kg/kW'},
                    }
          }
power = {'subelement_type':'PowerPhD',
         'params':{'generator_type':'PV',
                   'transmission_efficiency':0.9,
                   'cell_efficiency':0.175,
                   'cell_degradation':{'val':3.75,'units':'1/yr'},
                   'array_density':{'val':3.5, 'units':'kg/m**2'},
                   'discharge_depth':0.2,
                   'storage_specific_energy':{'val':30.,'units':'W*h/kg'},
                   'energy_transfer':'peak-tracking',
                   'low_array_degradation':False,
                   'orbit_period':{'val':2.,'units':'h'}, # low lunar orbit
                   'max_eclipse':{'val':0.5,'units':'h'}, # low lunar orbit
                   'ops_distance':{'val':1.,'units':'AU'}, # low lunar orbit
                   'mission_duration':{'val':0.5,'units':'yr'}
                  }
        }
structures = {'subelement_type':'StructuresPhD',
              'params':{'manned':False,
                        'truss':False,
                        'adapter':False,
                        'composite':False,
                        'A_de':-1.
                        }
             }
tanks = {'subelement_type':'TanksPhD',
         'params':{'num_fuel_tanks_mps':2,
                   'num_ox_tanks_mps':2,
                   'fuel_pressure_mps':{'val':40.,'units':'psi'},
                   'ox_pressure_mps':{'val':50.,'units':'psi'},
                   'ld_ratio_fuel_tanks_mps':1.,
                   'ld_ratio_ox_tanks_mps':1.,
                   'separator_type_mps':'pmd',
                   'num_fuel_tanks_rcs':1,
                   'num_ox_tanks_rcs':1,
                   'fuel_pressure_rcs':{'val':40.,'units':'psi'},
                   'ox_pressure_rcs':{'val':50.,'units':'psi'},
                   'ld_ratio_fuel_tanks_rcs':1.,
                   'ld_ratio_ox_tanks_rcs':1.,
                   'separator_type_rcs':'pmd',
                   'pressurant':'Helium',
                   'pressurant_pressure':{'val':6000.,'units':'psi'},
                   'num_tanks_pressurant':2,
                   'tank_ld_ratio_pressurant':1.0,
                   'material_strength':{'val':538.,'units':'MPa'},
                   'material_density':{'val':2685.,'units':'kg/m**3'},
                   'copv_pressurant_tank':False,
                   'composite_fuel_tanks_mps':False,
                   'composite_ox_tanks_mps':False,
                   'composite_fuel_tanks_rcs':False,
                   'composite_ox_tanks_rcs':False,
                   'ivfm':False,
                  }
        }
thermal = {'subelement_type':'ThermalPhD',
           'params':{'mli_layers_mps':60,
                     'mli_layers_rcs':60,
                     'active_cooling_mps':False,
                     'active_cooling_rcs':False,
                     'radiator_density':{'val':4.5,'units':'kg/m**2'},
                     'external_tanks':False,
                     'hi_efficiency_radiators':False,
                     'ops_distance':{'val':1.0,'units':'AU'},
                     'deep_space':False,
                     'orbit_alt':{'val':500.,'units':'km'}, # low lunar
                     'r_body':{'val':1737.,'units':'km'}, # moon
                     'T_body':{'val':270.,'units':'K'}, # moon
                     'albedo':0.12 # moon
                    }
          }
# fixed mass subsystem for payload
submass = {'subelement_type':'FixedMass','params':{'mass':{'val':1000.0,'units':'kg'}}}
stage = {'element_type':'Stage','subelement_list':[engines,power,structures,tanks,thermal,avionics],
         'params':{'auto_drop':False,'mps_reserve':3.,'rcs_reserve':3.,
                   'boiloff_model':'constant-rate','mga':20.0}}
s0 = deepcopy(stage)
s1 = deepcopy(stage)
s2 = deepcopy(stage)
# general payload element definitions
payload = {'element_type':'Payload','subelement_list':[submass],'params':{'auto_drop':False}}
DSH = deepcopy(payload); DSH['subelement_list'][0]['params']['mass'] = 20000.
ConRd_33 = deepcopy(payload); ConRd_33['subelement_list'][0]['params']['mass'] = 9000.
ConMOI_33 = deepcopy(payload); ConMOI_33['subelement_list'][0]['params']['mass'] = 2000.
ConEOI_33 = deepcopy(payload); ConEOI_33['subelement_list'][0]['params']['mass'] = 1000.
element_list = [s0,s1,s2,DSH,ConRd_33,ConMOI_33,ConEOI_33]
VEHICLE = {'element_list':element_list}
```

# APPENDIX E

# LAYERED PARETO FRONTS

A Pareto front is a set of undominated data points in a multi-objective space. Figure 89 illustrates the concept of Pareto front layers for a two-dimensional objective space where both objectives or minimized. For each layer, the set of design points are undominated by any design points of a higher layer. The layered Pareto front consists of the combination of all points in each layer up to and including the desired layer number. For instance, a three-layered Pareto front would consist of the combination of all the points on layers one, two, and three in Figure 89.

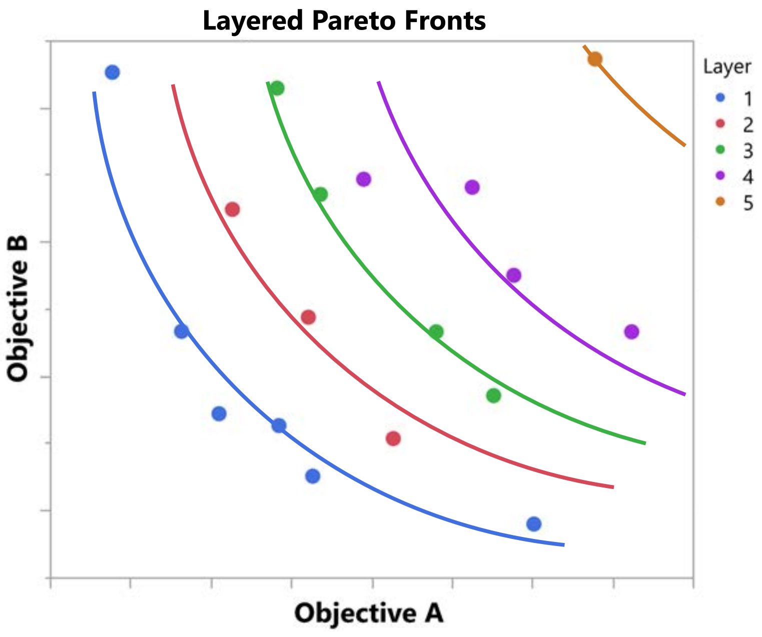


**Figure 89:** Notional Example of Pareto Front Layers

Each layer is obtained by excluding the previous layer from the objective space and evaluating a new Pareto front. This process of exclusion and reevaluation continues for the desired number of layers. This process may continue until all design points of the entire objective space are returned. The following JMP add-in was developed for the purpose of providing a simple interface for selecting these multi-layer Pareto fronts from a set of multi-objective data for this thesis.

## E.1 JSL Layred Pareto Front Analysis Script

```
//Layered Pareto Fronts
//By Douglas J. Trent
// douglas.trent@nasa.gov

Names Default To Here( 1 );

get_settings = function({cols},

        setwin = New Window("Layered Pareto Fronts for "||Char(dt << Get Name())||"",
                << modal,
                Panel Box("Select dominant high values",
                        Text Box("Check boxes to maximize."),
                        Text Box("Uncheck boxes to minimize."),
                        spacer box(size (275,5)),
                        cb_rol = Checkbox( cols )
                ),
                Panel Box("Options",
                        Lineup Box(
                                N Col( 3 ),
                                Text Box("Select number of (Pareto) Dominant layers:"),
                                nb = Number Edit Box( 1 , 3), nb << Set Increment(1),
                                sb = Spin Box(
                                        Function( {value},
                                                if(value >= 1,
                                                        nb << Increment( value ),
                                                value <= -1,
                                                        if(nb << Get() > 1, nb << Increment( value ) )
                                                )
                                        )
                                )
                        ),
                        H List Box(
                                Spacer Box(size(98,0)),
                                Text Box("Create subset data table:"),
                                cb_sub = Check Box( "" )
                        ),
                        H List Box(
                                Spacer Box(size(125,0)),
                                Text Box("Save script to table:"),
                                cb_scr = Check Box( "" )
                        ),
                        H List Box(
                                Spacer Box(size(93,0)),
                                Text Box("Hide non-dominant rows:"),
                                cb_h = Check Box( "" )
                        ),
                        H List Box(
                                Spacer Box(size(78,0)),
                                Text Box("Exclude non-dominant rows:"),
                                cb_e = Check Box( "" )
                        )
                ),
                Spacer Box(size(0,10)),
                H List Box(
                        Spacer Box(size(163,0)),
                        Button Box("OK",
                                b = 1;
                                l = nb << Get();
                                if(l < 1, l = 1);
                                s = cb_sub << Get();
                                sc = cb_scr << Get();
                                h = cb_h << Get();
                                e = cb_e << Get();
                                r = {};
                                for(i=1, i<=nitems(cols), i++,
                                        r[i] = cb_rol << Get(i);
                                );
                        ),
                        Button Box("Cancel",
                                b = 0;
                                l = 1;
                                s = 0;
                                r = {};
                                sc = 0;
                                h = 0;
                                e = 0
                        )
                )
        );

        return(evalList({b,l,s,r,sc,h,e}))

);

get_cols = function({dt},

        colwin = New Window("Layered Pareto Fronts for "||Char(dt << Get Name())||"",
                << modal,
                Text Box("Select columns for (Pareto) Dominant points"),
                Spacer Box(size(0,10)),
                columnList = Col List Box(dt,
                        all,
                        width(250),
                        nlines(30)
                ),
                Spacer Box(size(0,10)),
                H List Box(
                        Spacer Box(size(145,0)),
                        Button Box("OK",
                                c = columnList << Get Selected();
                                if(nitems(c)>0,
                                        b = 1,
                                        b = 2
                                )
                        ),
                        Button Box("Cancel",
                                b = 0;
                                c = {};
                                l = 1;
                                s = 0;
                                r = {};
                                sc = 0;
                                h = 0;
                                e = 0;
                        )
                )
        );

        if(b==1,
                {b,l,s,r,sc,h,e} = get_settings(c)
        );

        return(evalList({b,c,l,s,r,sc,h,e}));
);
```

```
Select Layered Dominant = function({cols={},roles={},layers=1,subset=0,hide=0,exclude=0},

        // get the data table
        dt = Current Data Table();

        // select columns
        if(cols != {} & nitems(roles) == nitems(cols) & layers > 0, button=1; scr=0;,
                {button,cols,layers,subset,roles,scr,hide,exclude} = get_cols(dt)
        );

        // parse user selection
        if(button>0 & nitems(cols) > 0,

                // get the included row
                rows = dt << Select Where( Excluded(Row State()) != 1 ) << Get Selected Rows();
                nrows = nrows(rows);
                StatusMsg("# of included rows: "||Char(nrows)||"");

                StatusMsg("performing lpf on "||Char(nitems(cols))||" columns");

                // run lpf
                pfs = {};
                for(i=1, i<=layers, i++,
                        StatusMsg("Calculating layer "||Char(i)||"");
                        dt << Select Dominant( {cols}, roles );
                        dt << Exclude;
                        dt << Label;
                        //lpf(cols, roles, dt);
                );

                // reset row states of lpfs
                dt << Select Where( Labeled(Row State()) == 1 ) << Exclude << Label;

                // create subset table of Pareto front
                if(subset > 0,
                        dt << Subset(Output Table(""||Char(layers)||"-Layered Pareto Front Analysis"),
                        ↪  Selected Rows, All Columns)
                );

                // hide all other rows
                if(hide > 0,
                        dt << Invert Row Selection << Hide;
                        dt << Invert Row Selection
                );

                // exclude all other rows
                if(exclude > 0,
                        dt << Invert Row Selection << Exclude;
                        dt << Invert Row Selection
                );

                // create table script
                if(scr == 1,
                        lpfscript = "
                        include(\!"$ADDIN_HOME(com.trent.lpf)\lpf.jsl\!");
                        cols = "||Char(cols)||";
                        roles = "||Char(roles)||";
                        layers = "||Char(layers)||";
                        subset = "||Char(subset)||";
                        hide = "||Char(hide)||";
                        exclude = "||Char(exclude)||";
                        Select Layered Dominant(cols,roles,layers,subset,hide,exclude);
                        ";
                        eval(parse("dt<<New Script(\!"LPFs\!","||lpfscript||");"));
                );

                StatusMsg("operation complete"),
        button == 0,

                StatusMsg("operation canceled by user"),

                StatusMsg("no columns selected, operation aborted");
                /*
                new window("Error Message",
                        <<modal,
                        VList Box(align("center"),
                                Text Box("No columns selected"),
                                Text Box("Operation aborted"),
                                Spacer Box(size(0,10)),
                                Button Box("OK")
                        )
                )
                */

        );
);
```

# APPENDIX F

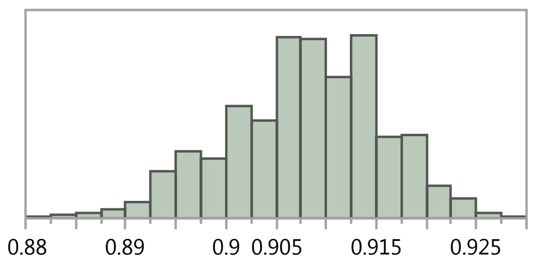
# EXPERIMENT 2 SIMILARITY DISTRIBUTION SUMMARY STATISTICS

The purpose of Experiment 2 is to determine if presenting results of individual architectures for these large spaces would prevent high-level design decisions from being studied. It is expected that high-level architecture trends will be difficult to observe when limiting the top results due to such large numbers of a single architecture type in the optimal objective space. The similarity metric described in Chapter 4.5.4 was utilized as a measure of likeness of alternatives. The distributions of this similarity metric of the alternatives for the top N design points were examined to reach conclusions with regard to Hypothesis 5.1. The following set of data supports the observations detailed in Chapter 5.1.3.

Data was collected for a variety of objective spaces: single-objective in cost, single-objective in mass, and multi-objective in cost and mass. This was due to the inconsistency between number of alternatives with number of Pareto front layers based on the defined objective space. For each of these objective spaces, varying-sized sets of top alternatives were examined for similarity of alternatives. The tables summarize the distribution statistics for each of the sets, while the graphs provide visualization of the distributions. These distributions of the top N alternative subsets can be compared to the distribution of similarity for all alternatives in Figure 90 to determine the quality of representation of the full set of alternatives provided by the top N subset distribution.

**Table 54:** LPF Layers and Number of Design Points

| | N | | |
|---|---|---|---|
| **# of Layers** | **Multi-Objective** | **Mass-Objective** | **Cost-Objective** |
| 1 | 2 | 2 | 2 |
| 2 | 6 | 4 | 4 |
| 3 | 10 | 6 | 6 |
| 4 | 16 | 8 | 8 |
| 5 | 24 | 10 | 10 |
| 10 | 68 | 20 | 20 |
| 20 | 226 | 40 | 40 |
| 30 | 418 | 60 | 62 |
| 40 | 830 | 80 | 88 |
| 50 | 1328 | 100 | 108 |
| 100 | 5090 | 200 | 218 |
| 200 | 17030 | 402 | 458 |
| 300 | 37651 | 604 | 708 |
| 400 | 76937 | 806 | 974 |
| 500 | 135508 | 1008 | 1230 |



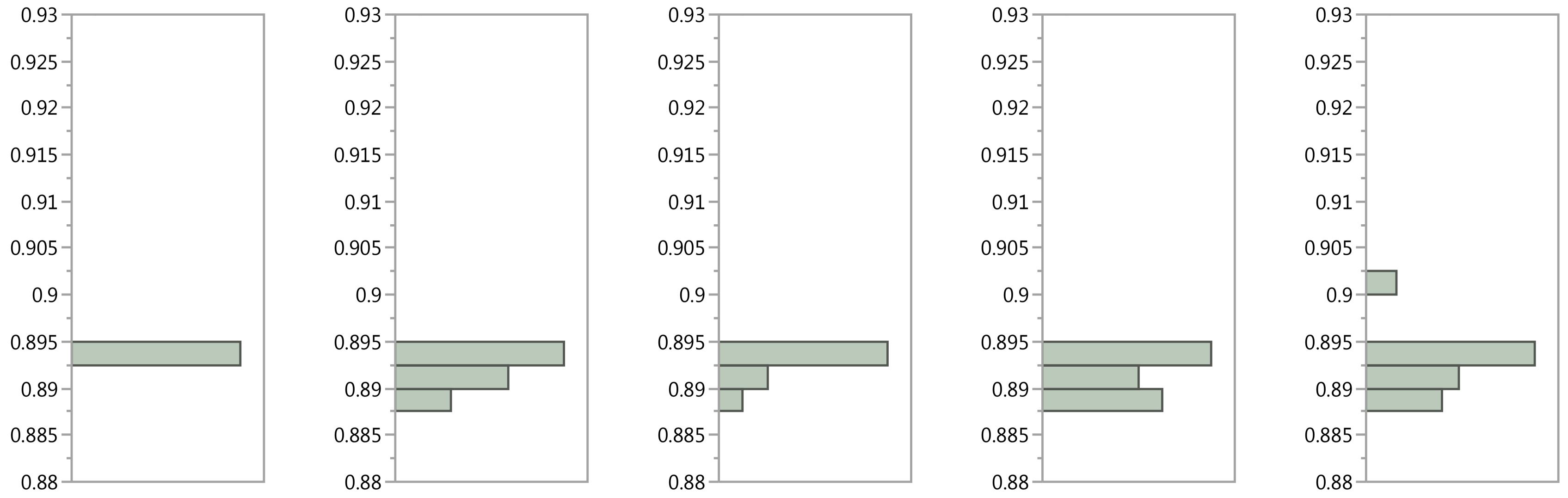
| **Quantiles** | |
|---|---|
| maximum | 0.928456 |
| minimum | 0.880507 |

| **Summary Statistics** | |
|---|---|
| Mean | 0.9079202 |
| Std Dev | 0.0074955 |
| Std Err Mean | 2.5095e-6 |
| Upper 95% Mean | 0.9079251 |
| Lower 95% Mean | 0.9079153 |
| N | 8921088 |

**Figure 90:** Full Objective Space Similarity Distribution

**Table 55:** Multi-Objective Similarity Distribution Summary

| N | Max | Min | Range | $\mu$ | $\sigma$ |
|---:|---:|---:|---:|---:|---:|
| 2 | 0.894712 | 0.893173 | 0.001539 | 0.8939425 | 0.0010882 |
| 6 | 0.894712 | 0.889739 | 0.004973 | 0.8922850 | 0.0017542 |
| 10 | 0.894712 | 0.889739 | 0.004973 | 0.8929480 | 0.0016448 |
| 16 | 0.894712 | 0.888201 | 0.006511 | 0.8918409 | 0.0020614 |
| 24 | 0.902106 | 0.888201 | 0.013905 | 0.8929012 | 0.0031756 |
| 68 | 0.902106 | 0.888201 | 0.013905 | 0.8944559 | 0.0043119 |
| 226 | 0.903339 | 0.886662 | 0.016677 | 0.8944350 | 0.0045575 |
| 418 | 0.903339 | 0.885123 | 0.018216 | 0.8940312 | 0.0045128 |
| 830 | 0.909622 | 0.885123 | 0.024499 | 0.8946365 | 0.0046238 |
| 1328 | 0.909622 | 0.885123 | 0.024499 | 0.8949155 | 0.0047745 |
| 5090 | 0.909622 | 0.883585 | 0.026037 | 0.8960166 | 0.0046291 |
| 17030 | 0.916179 | 0.882046 | 0.034133 | 0.8969876 | 0.0047947 |
| 37651 | 0.916395 | 0.882046 | 0.034349 | 0.8982201 | 0.0052000 |
| 76937 | 0.920666 | 0.882046 | 0.038620 | 0.8997035 | 0.0055028 |
| 135508 | 0.921899 | 0.882046 | 0.039853 | 0.9008529 | 0.0058279 |
| 8921088 | 0.928456 | 0.880507 | 0.047949 | 0.9079200 | 0.0074960 |



| Quantiles | |
|---|---|
| maximum | 0.894712 |
| minimum | 0.893173 |

| Summary Statistics | |
|---|---|
| Mean | 0.8939425 |
| Std Dev | 0.0010882 |
| Std Err Mean | 0.0007695 |
| Upper 95% Mean | 0.9037199 |
| Lower 95% Mean | 0.8841651 |
| N | 2 |

| Quantiles | |
|---|---|
| maximum | 0.894712 |
| minimum | 0.889739 |

| Summary Statistics | |
|---|---|
| Mean | 0.892285 |
| Std Dev | 0.0017542 |
| Std Err Mean | 0.0007161 |
| Upper 95% Mean | 0.8941259 |
| Lower 95% Mean | 0.8904441 |
| N | 6 |

| Quantiles | |
|---|---|
| maximum | 0.894712 |
| minimum | 0.889739 |

| Summary Statistics | |
|---|---|
| Mean | 0.892948 |
| Std Dev | 0.0016448 |
| Std Err Mean | 0.0005201 |
| Upper 95% Mean | 0.8941246 |
| Lower 95% Mean | 0.8917714 |
| N | 10 |

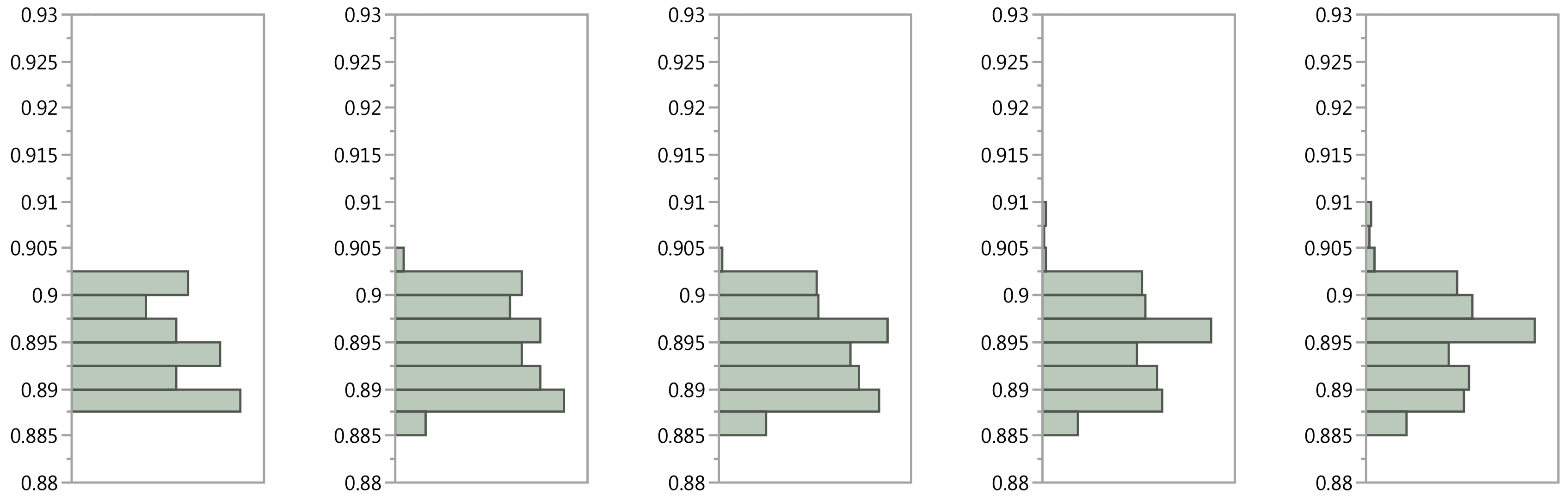
| Quantiles | |
|---|---|
| maximum | 0.894712 |
| minimum | 0.888201 |

| Summary Statistics | |
|---|---|
| Mean | 0.8918409 |
| Std Dev | 0.0020614 |
| Std Err Mean | 0.0005154 |
| Upper 95% Mean | 0.8929393 |
| Lower 95% Mean | 0.8907424 |
| N | 16 |

| Quantiles | |
|---|---|
| maximum | 0.902106 |
| minimum | 0.888201 |

| Summary Statistics | |
|---|---|
| Mean | 0.8929012 |
| Std Dev | 0.0031756 |
| Std Err Mean | 0.0006482 |
| Upper 95% Mean | 0.8942422 |
| Lower 95% Mean | 0.8915603 |
| N | 24 |

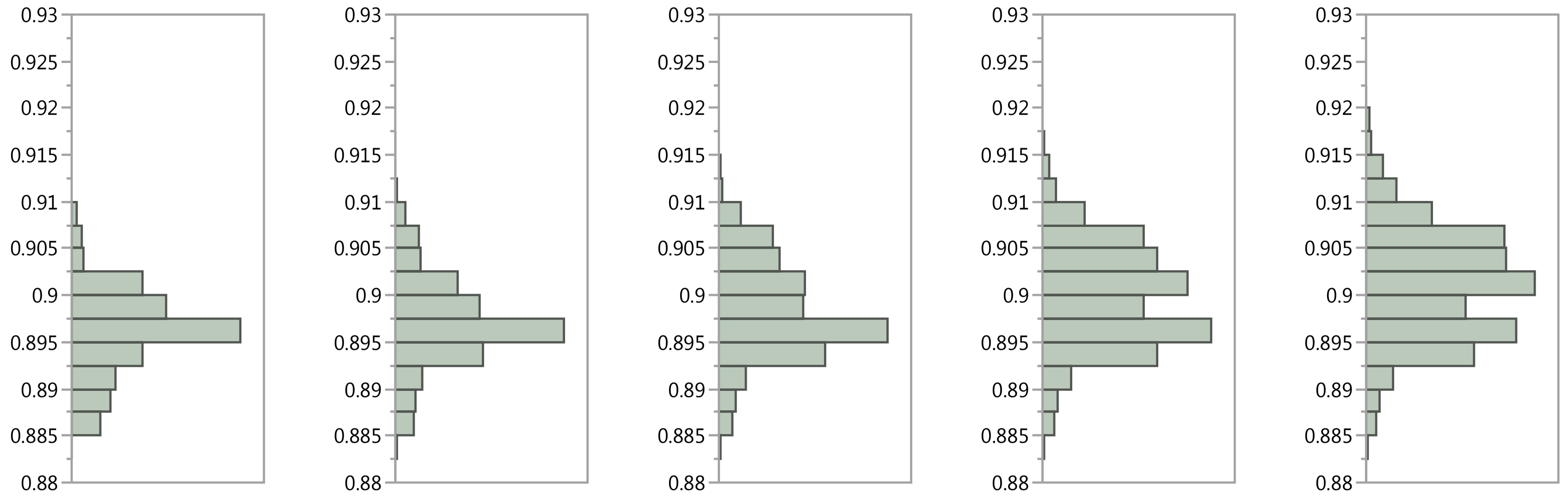
**Figure 91:** Multi-Objective Similarity Distributions, 1 to 5 Layered Pareto Front in Steps of 1



| Quantiles | | | | | |
|---|---|---|---|---|---|
| maximum | 0.902106 | 0.903339 | 0.903339 | 0.909622 | 0.909622 |
| minimum | 0.888201 | 0.886662 | 0.885123 | 0.885123 | 0.885123 |

| Summary Statistics | | | | | |
|---|---|---|---|---|---|
| Mean | 0.8944559 | 0.894435 | 0.8940312 | 0.8946365 | 0.8949155 |
| Std Dev | 0.0043119 | 0.0045575 | 0.0045128 | 0.0046238 | 0.0047745 |
| Std Err Mean | 0.0005229 | 0.0003032 | 0.0002207 | 0.0001605 | 0.000131 |
| Upper 95% Mean | 0.8954995 | 0.8950325 | 0.8944651 | 0.8949515 | 0.8951725 |
| Lower 95% Mean | 0.8934122 | 0.8938376 | 0.8935973 | 0.8943215 | 0.8946584 |
| N | 68 | 226 | 418 | 830 | 1328 |

**Figure 92:** Multi-Objective Similarity Distributions, 10 to 50 Layered Pareto Front in Steps of 10



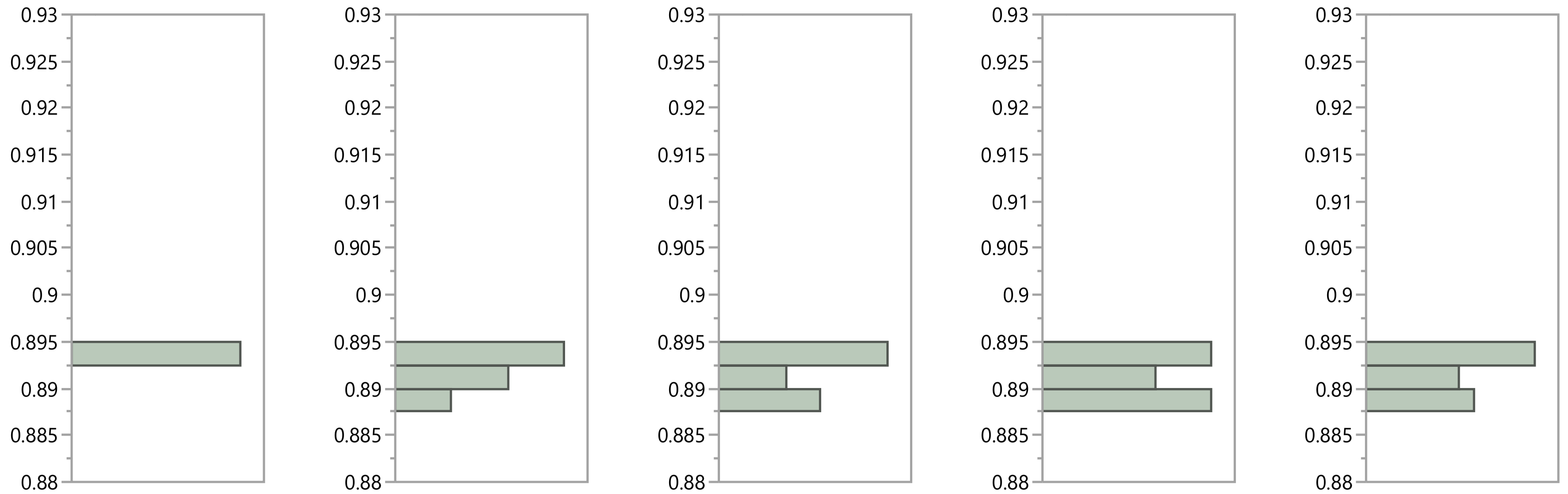
| Quantiles | | | | | |
|---|---|---|---|---|---|
| maximum | 0.909622 | 0.916179 | 0.916395 | 0.920666 | 0.921899 |
| minimum | 0.883585 | 0.882046 | 0.882046 | 0.882046 | 0.882046 |

| Summary Statistics | | | | | |
|---|---|---|---|---|---|
| Mean | 0.8960166 | 0.8969876 | 0.8982201 | 0.8997035 | 0.9008529 |
| Std Dev | 0.0046291 | 0.0047947 | 0.0052 | 0.0055028 | 0.0058279 |
| Std Err Mean | 6.4884e-5 | 3.6741e-5 | 0.0000268 | 1.9839e-5 | 1.5832e-5 |
| Upper 95% Mean | 0.8961438 | 0.8970596 | 0.8982726 | 0.8997424 | 0.900884 |
| Lower 95% Mean | 0.8958894 | 0.8969156 | 0.8981675 | 0.8996646 | 0.9008219 |
| N | 5090 | 17030 | 37651 | 76937 | 135508 |

**Figure 93:** Multi-Objective Similarity Distributions, 100 to 500 Layered Pareto Front in Steps of 100

**Table 56:** Mass-Objective Similarity Distribution Summary

| N | Max | Min | Range | $\mu$ | $\sigma$ |
|---:|---:|---:|---:|---:|---:|
| 2 | 0.894712 | 0.893173 | 0.001539 | 0.8939425 | 0.0010882 |
| 6 | 0.894712 | 0.889739 | 0.004973 | 0.8922850 | 0.0017542 |
| 10 | 0.894712 | 0.888201 | 0.006511 | 0.8919535 | 0.0022143 |
| 16 | 0.894712 | 0.888201 | 0.006511 | 0.8914563 | 0.0020993 |
| 24 | 0.894712 | 0.888201 | 0.006511 | 0.8918706 | 0.0020328 |
| 68 | 0.902106 | 0.888201 | 0.013905 | 0.8949466 | 0.0042156 |
| 226 | 0.902106 | 0.885123 | 0.016983 | 0.8933503 | 0.0045252 |
| 418 | 0.902228 | 0.885123 | 0.017105 | 0.8936429 | 0.0044170 |
| 830 | 0.909622 | 0.885123 | 0.024499 | 0.8939707 | 0.0048318 |
| 1328 | 0.909622 | 0.883585 | 0.026037 | 0.8944469 | 0.0051703 |
| 5090 | 0.909622 | 0.883585 | 0.026037 | 0.8959993 | 0.0050216 |
| 17030 | 0.916179 | 0.882046 | 0.034133 | 0.8975037 | 0.0052876 |
| 37651 | 0.919829 | 0.882046 | 0.037783 | 0.8987812 | 0.0056004 |
| 76937 | 0.920666 | 0.882046 | 0.038620 | 0.9001654 | 0.0059191 |
| 135508 | 0.927223 | 0.882046 | 0.045177 | 0.9013363 | 0.0060132 |
| 8921088 | 0.928456 | 0.880507 | 0.047949 | 0.9079200 | 0.0074960 |



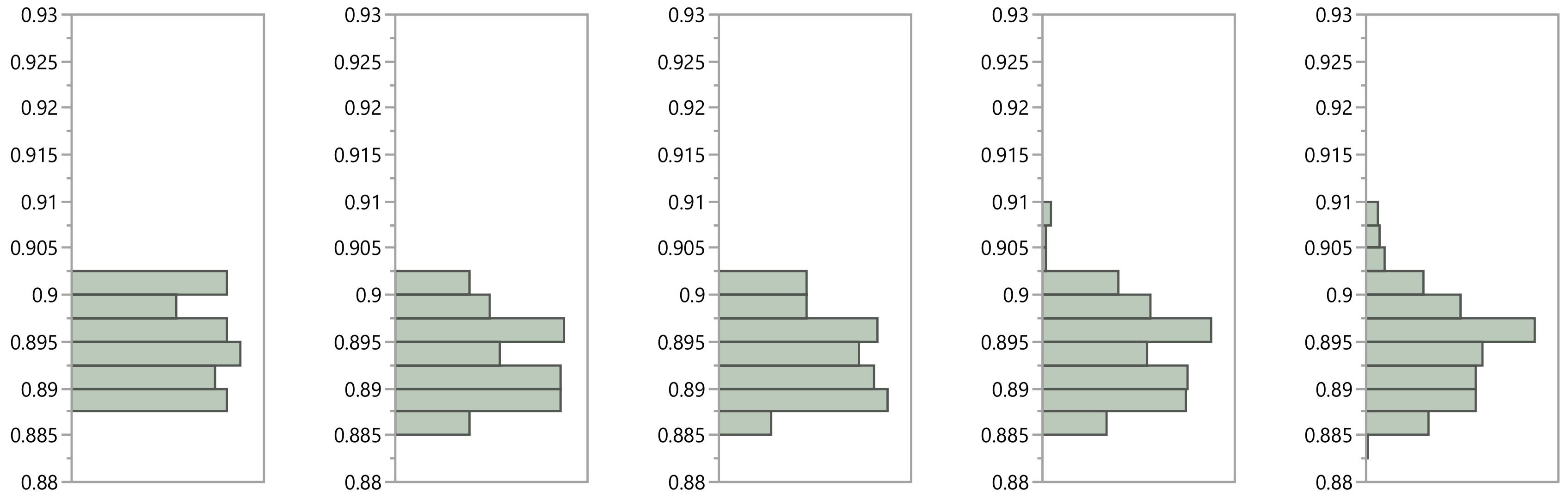
**Quantiles**

| | N=2 | N=6 | N=10 | N=16 | N=24 |
|---|---|---|---|---|---|
| maximum | 0.894712 | 0.894712 | 0.894712 | 0.894712 | 0.894712 |
| minimum | 0.893173 | 0.889739 | 0.888201 | 0.888201 | 0.888201 |

**Summary Statistics**

| | | | | | |
|---|---|---|---|---|---|
| Mean | 0.8939425 | 0.892285 | 0.8919535 | 0.8914563 | 0.8918706 |
| Std Dev | 0.0010882 | 0.0017542 | 0.0022143 | 0.0020993 | 0.0020328 |
| Std Err Mean | 0.0007695 | 0.0007161 | 0.0007002 | 0.0005248 | 0.0004149 |
| Upper 95% Mean | 0.9037199 | 0.8941259 | 0.8935375 | 0.8925749 | 0.892729 |
| Lower 95% Mean | 0.8841651 | 0.8904441 | 0.8903695 | 0.8903376 | 0.8910123 |
| N | 2 | 6 | 10 | 16 | 24 |

**Figure 94:** Mass-Objective Similarity Distributions, N=2, 6, 10, 16, 24



**Quantiles**

| | |
|---|---|
| maximum | 0.902106 |
| minimum | 0.888201 |

**Summary Statistics**

| | |
|---|---|
| Mean | 0.8949466 |
| Std Dev | 0.0042156 |
| Std Err Mean | 0.0005112 |
| Upper 95% Mean | 0.895967 |
| Lower 95% Mean | 0.8939262 |
| N | 68 |

**Quantiles**

| | |
|---|---|
| maximum | 0.902106 |
| minimum | 0.885123 |

**Summary Statistics**

| | |
|---|---|
| Mean | 0.8933503 |
| Std Dev | 0.0045252 |
| Std Err Mean | 0.000301 |
| Upper 95% Mean | 0.8939435 |
| Lower 95% Mean | 0.8927572 |
| N | 226 |

**Quantiles**

| | |
|---|---|
| maximum | 0.902228 |
| minimum | 0.885123 |

**Summary Statistics**

| | |
|---|---|
| Mean | 0.8936429 |
| Std Dev | 0.004417 |
| Std Err Mean | 0.000216 |
| Upper 95% Mean | 0.8940675 |
| Lower 95% Mean | 0.8932182 |
| N | 418 |

**Quantiles**

| | |
|---|---|
| maximum | 0.909622 |
| minimum | 0.885123 |

**Summary Statistics**

| | |
|---|---|
| Mean | 0.8939707 |
| Std Dev | 0.0048318 |
| Std Err Mean | 0.0001677 |
| Upper 95% Mean | 0.8942999 |
| Lower 95% Mean | 0.8936415 |
| N | 830 |

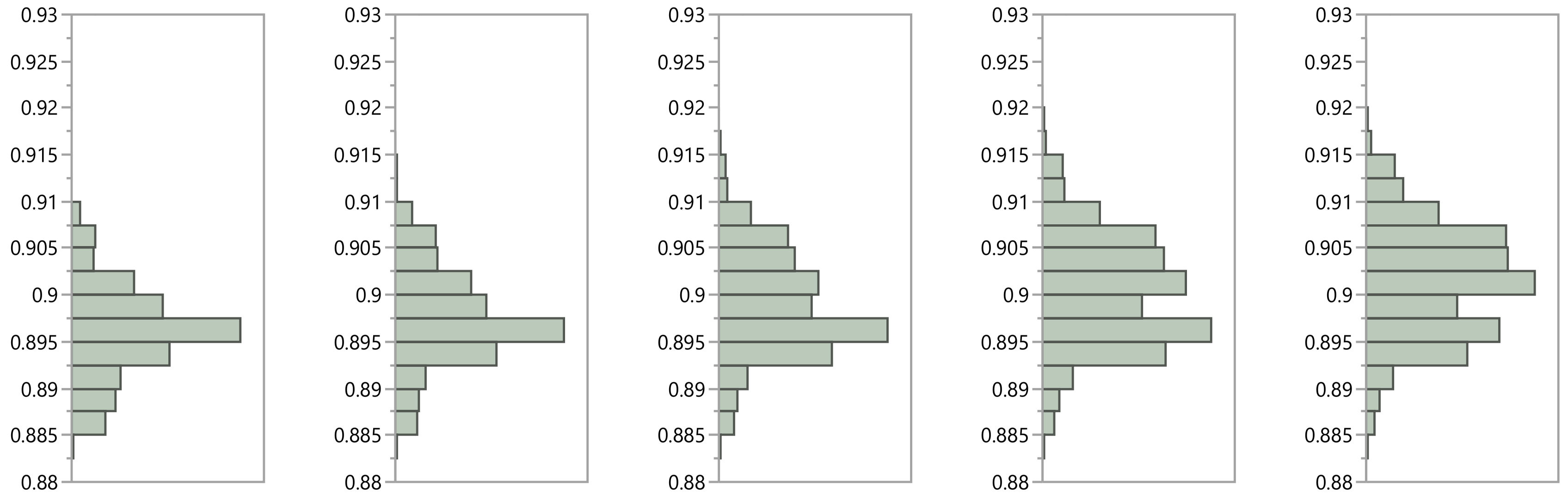
**Quantiles**

| | |
|---|---|
| maximum | 0.909622 |
| minimum | 0.883585 |

**Summary Statistics**

| | |
|---|---|
| Mean | 0.8944469 |
| Std Dev | 0.0051703 |
| Std Err Mean | 0.0001419 |
| Upper 95% Mean | 0.8947252 |
| Lower 95% Mean | 0.8941686 |
| N | 1328 |

**Figure 95:** Mass-Objective Similarity Distributions, N=68, 226, 418, 830, 1328



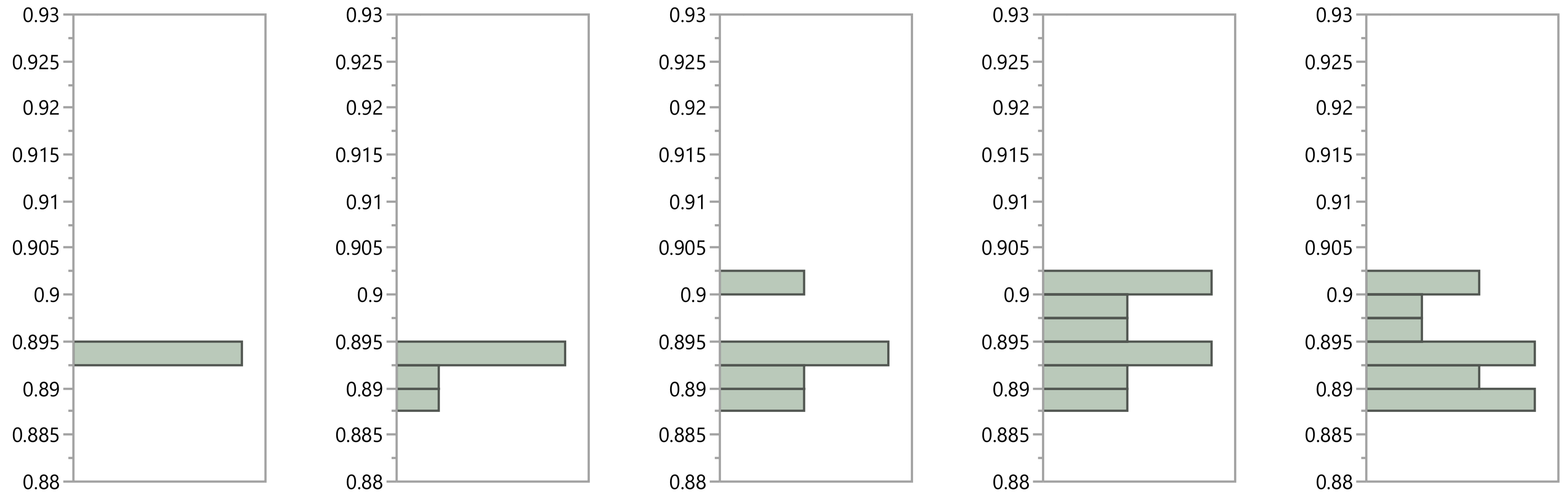
| Quantiles | | | | | |
|---|---|---|---|---|---|
| maximum | 0.909622 | 0.916179 | 0.919829 | 0.920666 | 0.927223 |
| minimum | 0.883585 | 0.882046 | 0.882046 | 0.882046 | 0.882046 |

| Summary Statistics | | | | | |
|---|---|---|---|---|---|
| Mean | 0.8959993 | 0.8975037 | 0.8987812 | 0.9001654 | 0.9013363 |
| Std Dev | 0.0050216 | 0.0052876 | 0.0056004 | 0.0059191 | 0.0060132 |
| Std Err Mean | 7.0385e-5 | 4.0518e-5 | 2.8862e-5 | 2.134e-5 | 1.6335e-5 |
| Upper 95% Mean | 0.8961373 | 0.8975831 | 0.8988378 | 0.9002072 | 0.9013683 |
| Lower 95% Mean | 0.8958613 | 0.8974243 | 0.8987247 | 0.9001235 | 0.9013042 |
| N | 5090 | 17030 | 37651 | 76937 | 135508 |

**Figure 96:** Mass-Objective Similarity Distributions, N=5090, 17030, 37651, 76937, 135508

**Table 57:** Cost-Objective Similarity Distribution Summary

| N | Max | Min | Range | $\mu$ | $\sigma$ |
|---:|---:|---:|---:|---:|---:|
| 2 | 0.894712 | 0.893173 | 0.001539 | 0.8939425 | 0.0010882 |
| 6 | 0.894712 | 0.889739 | 0.004973 | 0.8922850 | 0.0017542 |
| 10 | 0.894712 | 0.889739 | 0.004973 | 0.8929480 | 0.0016448 |
| 16 | 0.894712 | 0.888201 | 0.006511 | 0.8918409 | 0.0020614 |
| 24 | 0.902106 | 0.888201 | 0.013905 | 0.8929012 | 0.0031756 |
| 68 | 0.902106 | 0.888201 | 0.013905 | 0.8944559 | 0.0043119 |
| 226 | 0.903339 | 0.886662 | 0.016677 | 0.8944350 | 0.0045575 |
| 418 | 0.903339 | 0.885123 | 0.018216 | 0.8940312 | 0.0045128 |
| 830 | 0.909622 | 0.885123 | 0.024499 | 0.8946365 | 0.0046238 |
| 1328 | 0.909622 | 0.885123 | 0.024499 | 0.8949155 | 0.0047745 |
| 5090 | 0.909622 | 0.883585 | 0.026037 | 0.8960166 | 0.0046291 |
| 17030 | 0.916179 | 0.882046 | 0.034133 | 0.8969876 | 0.0047947 |
| 37651 | 0.916395 | 0.882046 | 0.034349 | 0.8982201 | 0.0052000 |
| 76937 | 0.920666 | 0.882046 | 0.038620 | 0.8997035 | 0.0055028 |
| 135508 | 0.921899 | 0.882046 | 0.039853 | 0.9008529 | 0.0058279 |
| 8921088 | 0.928456 | 0.880507 | 0.047949 | 0.9079200 | 0.0074960 |



| **Quantiles** | |
|---|---|
| maximum | 0.894712 |
| minimum | 0.893173 |

| **Summary Statistics** | |
|---|---|
| Mean | 0.8939425 |
| Std Dev | 0.0010882 |
| Std Err Mean | 0.0007695 |
| Upper 95% Mean | 0.9037199 |
| Lower 95% Mean | 0.8841651 |
| N | 2 |

| **Quantiles** | |
|---|---|
| maximum | 0.894712 |
| minimum | 0.889739 |

| **Summary Statistics** | |
|---|---|
| Mean | 0.8927978 |
| Std Dev | 0.0019635 |
| Std Err Mean | 0.0008016 |
| Upper 95% Mean | 0.8948584 |
| Lower 95% Mean | 0.8907373 |
| N | 6 |

| **Quantiles** | |
|---|---|
| maximum | 0.902106 |
| minimum | 0.889739 |

| **Summary Statistics** | |
|---|---|
| Mean | 0.8940478 |
| Std Dev | 0.004247 |
| Std Err Mean | 0.001343 |
| Upper 95% Mean | 0.8970859 |
| Lower 95% Mean | 0.8910097 |
| N | 10 |

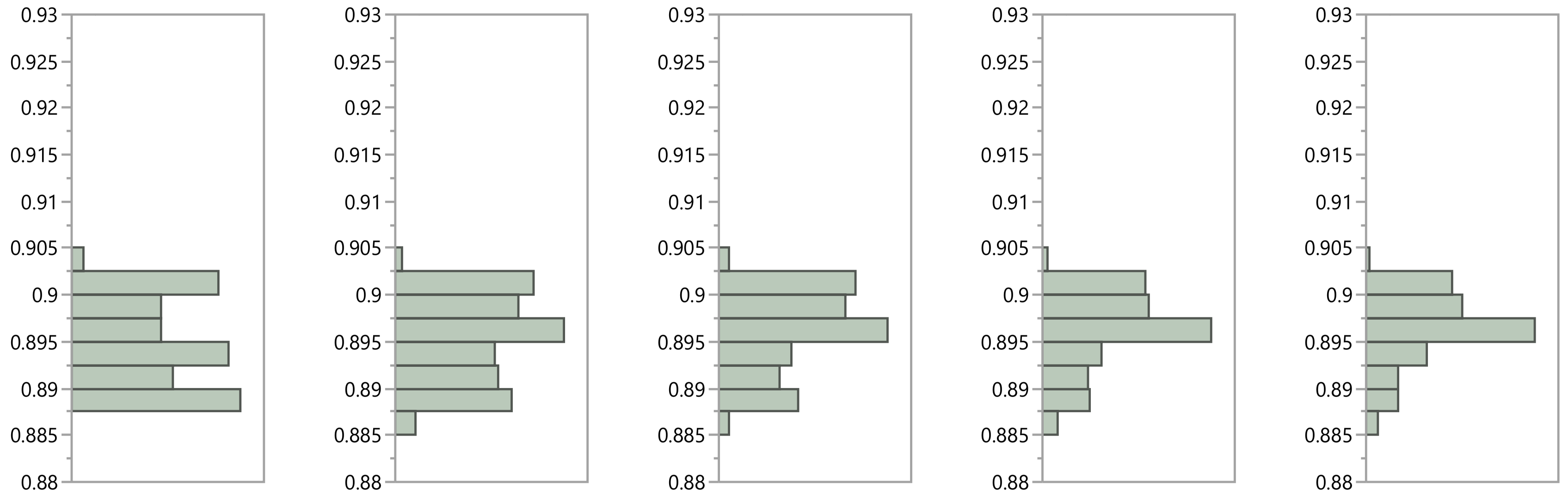
| **Quantiles** | |
|---|---|
| maximum | 0.902106 |
| minimum | 0.889739 |

| **Summary Statistics** | |
|---|---|
| Mean | 0.8959229 |
| Std Dev | 0.0042846 |
| Std Err Mean | 0.0010711 |
| Upper 95% Mean | 0.898206 |
| Lower 95% Mean | 0.8936398 |
| N | 16 |

| **Quantiles** | |
|---|---|
| maximum | 0.902106 |
| minimum | 0.888201 |

| **Summary Statistics** | |
|---|---|
| Mean | 0.8941776 |
| Std Dev | 0.0044227 |
| Std Err Mean | 0.0009028 |
| Upper 95% Mean | 0.8960451 |
| Lower 95% Mean | 0.89231 |
| N | 24 |

**Figure 97:** Mass-Objective Similarity Distributions, N=2, 6, 10, 16, 24



**Quantiles** (N = 68)

| | |
|---|---|
| maximum | 0.903339 |
| minimum | 0.888201 |

**Summary Statistics**

| | |
|---|---|
| Mean | 0.8949373 |
| Std Dev | 0.0045785 |
| Std Err Mean | 0.0005552 |
| Upper 95% Mean | 0.8960456 |
| Lower 95% Mean | 0.8938291 |
| N | 68 |

**Quantiles**

| | |
|---|---|
| maximum | 0.903339 |
| minimum | 0.886662 |

**Summary Statistics**

| | |
|---|---|
| Mean | 0.8953157 |
| Std Dev | 0.004464 |
| Std Err Mean | 0.0002969 |
| Upper 95% Mean | 0.8959009 |
| Lower 95% Mean | 0.8947306 |
| N | 226 |

**Quantiles**

| | |
|---|---|
| maximum | 0.903339 |
| minimum | 0.886662 |

**Summary Statistics**

| | |
|---|---|
| Mean | 0.8961894 |
| Std Dev | 0.0042232 |
| Std Err Mean | 0.0002066 |
| Upper 95% Mean | 0.8965955 |
| Lower 95% Mean | 0.8957834 |
| N | 418 |

**Quantiles**

| | |
|---|---|
| maximum | 0.903339 |
| minimum | 0.885123 |

**Summary Statistics**

| | |
|---|---|
| Mean | 0.8961976 |
| Std Dev | 0.0040759 |
| Std Err Mean | 0.0001415 |
| Upper 95% Mean | 0.8964753 |
| Lower 95% Mean | 0.8959199 |
| N | 830 |

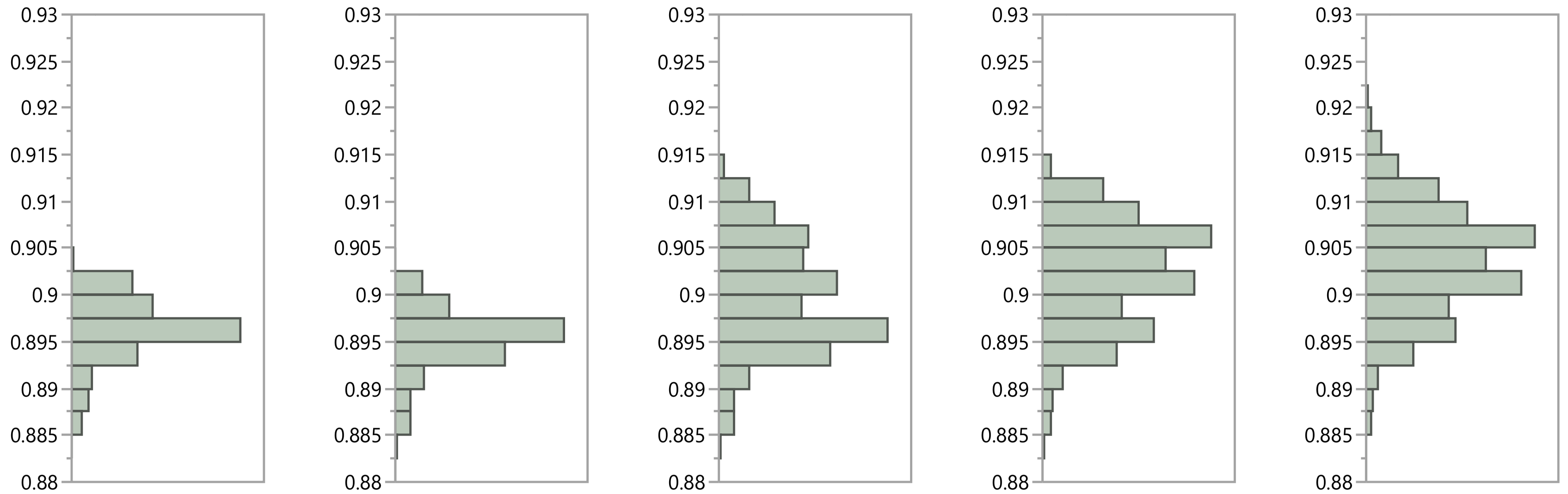
**Quantiles**

| | |
|---|---|
| maximum | 0.903339 |
| minimum | 0.885123 |

**Summary Statistics**

| | |
|---|---|
| Mean | 0.8963391 |
| Std Dev | 0.0038391 |
| Std Err Mean | 0.0001053 |
| Upper 95% Mean | 0.8965458 |
| Lower 95% Mean | 0.8961324 |
| N | 1328 |

**Figure 98:** Mass-Objective Similarity Distributions, N=68, 226, 418, 830, 1328



**Quantiles**

| | N=5090 | N=17030 | N=37651 | N=76937 | N=135508 |
|---|---|---|---|---|---|
| maximum | 0.903339 | 0.903339 | 0.914383 | 0.914383 | 0.921899 |
| minimum | 0.883585 | 0.882046 | 0.882046 | 0.882046 | 0.882046 |

**Summary Statistics**

| | | | | | |
|---|---|---|---|---|---|
| Mean | 0.8963583 | 0.8952793 | 0.8996781 | 0.9019521 | 0.9033841 |
| Std Dev | 0.0034665 | 0.0034553 | 0.0058658 | 0.0056648 | 0.0059171 |
| Std Err Mean | 4.8588e-5 | 2.6478e-5 | 3.023e-5 | 2.0423e-5 | 1.6074e-5 |
| Upper 95% Mean | 0.8964535 | 0.8953312 | 0.8997374 | 0.9019921 | 0.9034157 |
| Lower 95% Mean | 0.896263 | 0.8952274 | 0.8996189 | 0.9019121 | 0.9033526 |
| N | 5090 | 17030 | 37651 | 76937 | 135508 |

**Figure 99:** Mass-Objective Similarity Distributions, N=5090, 17030, 37651, 76937, 135508

# APPENDIX G

# EXPERIMENT 3 PORTFOLIO DISTRIBUTION DATA

The purpose of experiment 3 is to determine how parameters on which to group sets of architectures into portfolios affect the resulting figures of merit of those portfolios. Hypothesis 5.2 claims that the size of the portfolio is related to the variation of portfolio aggregate metrics, as well as the variation of various metrics between alternatives within a portfolio. Chapter 5.1.4 observed this hypothesis to be false, supported by observations in the variance of various objective parameters for different grouping criteria. This appendix provides detailed data for the distribution statistics of the portfolios formed by these grouping criteria supporting the observations detailed in Chapter 5.1.4.

**Table 58:** Mission Portfolios Distribution Summary Statistics

| **Description** | **N** | $\mu\ m_{gross}$ **(kg)** | $\sigma\ m_{gross}$ **(kg)** | $\mu\ C_{gross}$ **(MYr)** | $\sigma\ C_{gross}$ **(MYr)** | $\mu\ PMF$ | $\sigma\ PMF$ | $\mu\ S$ | $\sigma\ S$ |
|---|---|---|---|---|---|---|---|---|---|
| Long Stay | 3162889 | 3.856e+4 | 5.339e+8 | 4.344e+4 | 5.288e+8 | 6.567e-1 | 6.325e-3 | 9.083e-1 | 6.787e-5 |
| Short Stay | 3373681 | 3.777e+4 | 5.293e+8 | 4.399e+4 | 5.195e+8 | 6.515e-1 | 7.154e-3 | 9.100e-1 | 6.653e-5 |
| Large Correction | 3249608 | 3.976e+4 | 5.714e+8 | 4.401e+4 | 5.312e+8 | 6.626e-1 | 6.468e-3 | 9.084e-1 | 6.716e-5 |
| Small Correction | 3286962 | 3.656e+4 | 4.874e+8 | 4.343e+4 | 5.168e+8 | 6.455e-1 | 6.902e-3 | 9.099e-1 | 6.750e-5 |
| Long Large | 1574764 | 4.028e+4 | 5.777e+8 | 4.379e+4 | 5.382e+8 | 6.654e-1 | 6.046e-3 | 9.075e-1 | 6.704e-5 |
| Long Small | 1588125 | 3.685e+4 | 4.846e+8 | 4.309e+4 | 5.192e+8 | 6.481e-1 | 6.453e-3 | 9.090e-1 | 6.748e-5 |
| Short Large | 1674844 | 3.926e+4 | 5.649e+8 | 4.422e+4 | 5.245e+8 | 6.600e-1 | 6.851e-3 | 9.092e-1 | 6.579e-5 |
| Short Small | 1698837 | 3.630e+4 | 4.899e+8 | 4.376e+4 | 5.144e+8 | 6.431e-1 | 7.310e-3 | 9.108e-1 | 6.602e-5 |

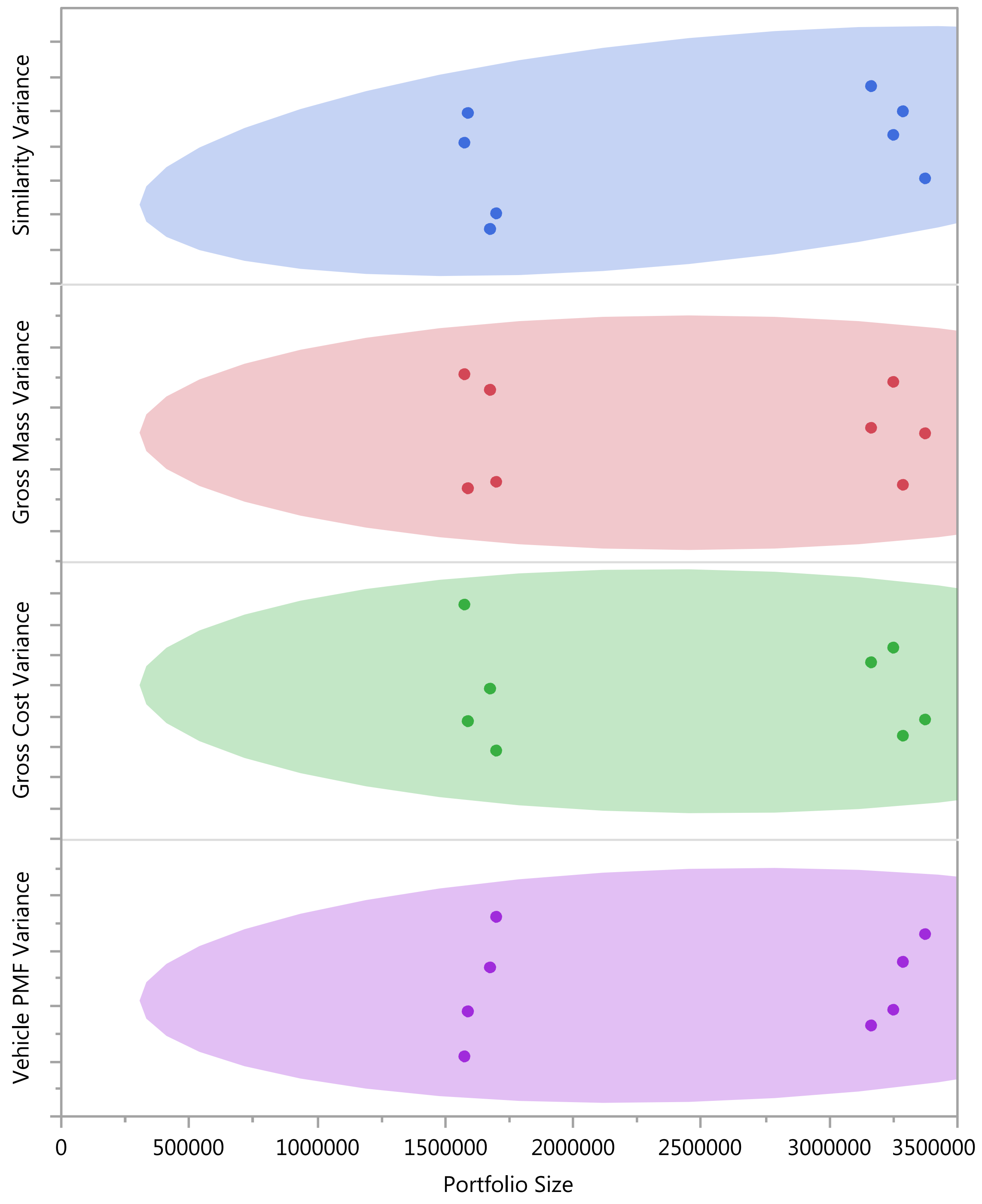


**Figure 100:** Correlation in Portfolio Size vs Objective Metric Variance for Mission-based Portfolios

**Table 59:** Technology Portfolios Distribution Summary Statistics

| **Description** | **N** | $\mu\, m_{gross}$ **(kg)** | $\sigma\, m_{gross}$ **(kg)** | $\mu\, C_{gross}$ **(MYr)** | $\sigma\, C_{gross}$ **(MYr)** | $\mu\, PMF$ | $\sigma\, PMF$ | $\mu\, S$ | $\sigma\, S$ |
|---|---|---|---|---|---|---|---|---|---|
| No Techs | 708506 | 3.894e+4 | 5.426e+8 | 4.009e+4 | 4.375e+8 | 6.612e-1 | 5.899e-3 | 9.111e-1 | 6.237e-5 |
| Wireless Sensors Only | 708554 | 3.890e+4 | 5.424e+8 | 4.417e+4 | 5.216e+8 | 6.613e-1 | 5.903e-3 | 9.099e-1 | 6.239e-5 |
| Composite Structures Only | 719846 | 3.715e+4 | 5.030e+8 | 4.325e+4 | 4.869e+8 | 6.661e-1 | 5.907e-3 | 9.099e-1 | 6.259e-5 |
| Composite Tanks only | 709442 | 3.857e+4 | 5.335e+8 | 4.402e+4 | 5.159e+8 | 6.617e-1 | 5.892e-3 | 9.099e-1 | 6.248e-5 |
| Active Cryocooling Only | 768338 | 3.861e+4 | 5.340e+8 | 4.634e+4 | 5.875e+8 | 6.309e-1 | 8.800e-3 | 9.100e-1 | 6.091e-5 |
| Integrated MPS/RCS Only | 708338 | 3.924e+4 | 5.510e+8 | 4.104e+4 | 4.385e+8 | 6.612e-1 | 5.873e-3 | 9.099e-1 | 6.231e-5 |
| Low Leak Valves Only | 709001 | 3.843e+4 | 5.431e+8 | 4.336e+4 | 5.345e+8 | 6.571e-1 | 6.026e-3 | 9.099e-1 | 6.244e-5 |
| High Capacity Batteries Only | 708817 | 3.859e+4 | 5.434e+8 | 4.188e+4 | 4.808e+8 | 6.616e-1 | 5.966e-3 | 9.099e-1 | 6.240e-5 |
| All Techs | 795728 | 3.527e+4 | 4.850e+8 | 4.854e+4 | 6.369e+8 | 6.299e-1 | 8.533e-3 | 9.027e-1 | 6.096e-5 |
| One Tech | 5032336 | 3.850e+4 | 5.360e+8 | 4.347e+4 | 5.128e+8 | 6.568e-1 | 6.493e-3 | 9.099e-1 | 6.221e-5 |
| Power Techs | 1504545 | 3.683e+4 | 5.152e+8 | 4.540e+4 | 5.744e+8 | 6.448e-1 | 7.575e-3 | 9.061e-1 | 7.435e-5 |
| Engine Techs | 1504729 | 3.676e+4 | 5.148e+8 | 4.610e+4 | 5.953e+8 | 6.427e-1 | 7.536e-3 | 9.061e-1 | 7.437e-5 |
| Tank Techs | 2213508 | 3.760e+4 | 5.248e+8 | 4.469e+4 | 5.443e+8 | 6.501e-1 | 7.065e-3 | 9.073e-1 | 7.363e-5 |
| Structures Techs | 1515574 | 3.616e+4 | 4.944e+8 | 4.602e+4 | 5.726e+8 | 6.470e-1 | 7.612e-3 | 9.062e-1 | 7.465e-5 |
| Avionics Techs | 1504282 | 3.698e+4 | 5.153e+8 | 4.648e+4 | 5.873e+8 | 6.447e-1 | 7.540e-3 | 9.061e-1 | 7.434e-5 |
| Thermal Techs | 1564066 | 3.691e+4 | 5.118e+8 | 4.746e+4 | 6.138e+8 | 6.304e-1 | 8.664e-3 | 9.063e-1 | 7.431e-5 |

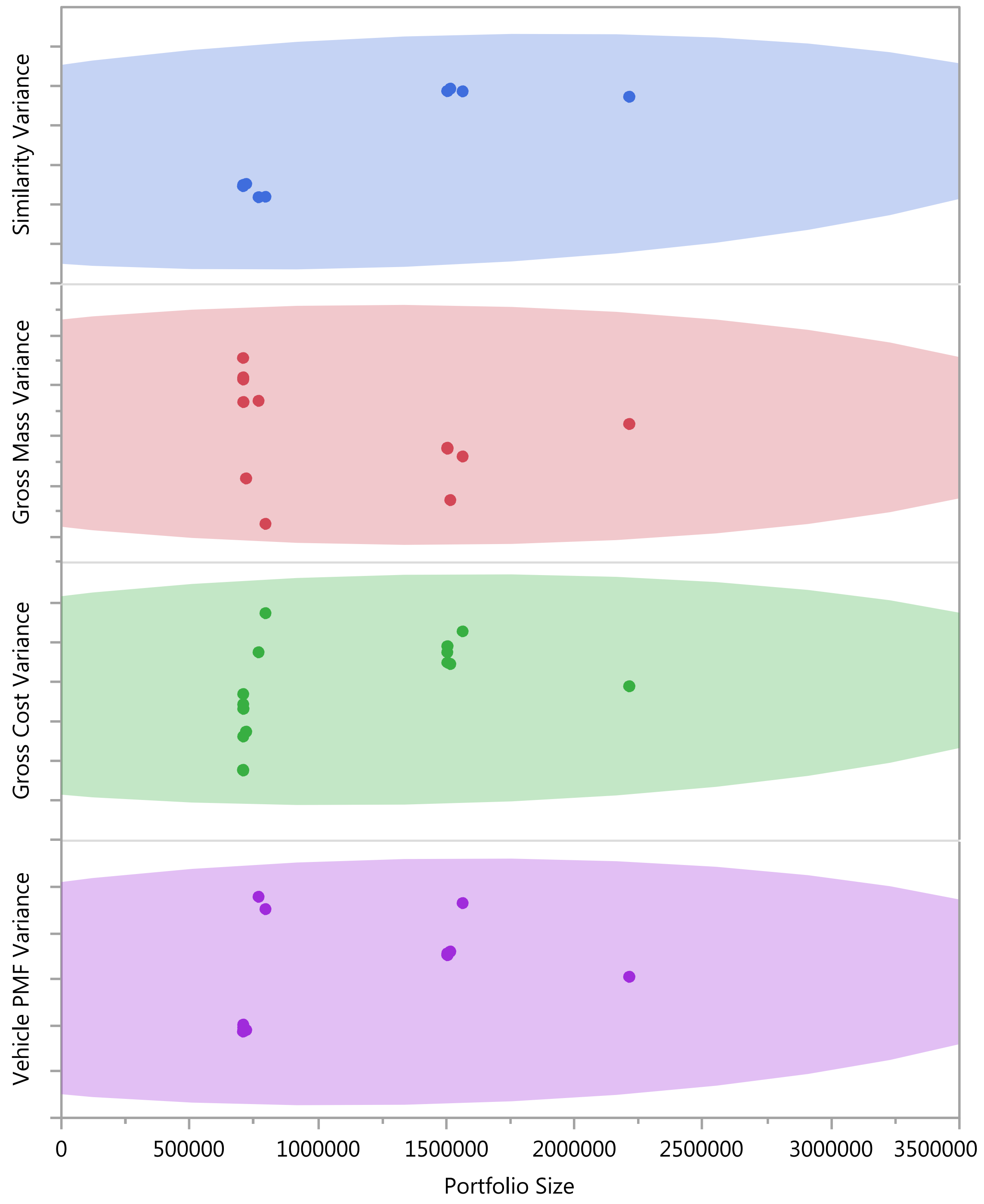


**Figure 101:** Correlation in Portfolio Size vs Objective Metric Variance for Technology-based Portfolios

**Table 60:** Vehicle Portfolios Distribution Summary Statistics

| Description | N | $\mu\, m_{gross}$ (kg) | $\sigma\, m_{gross}$ (kg) | $\mu\, C_{gross}$ (MYr) | $\sigma\, C_{gross}$ (MYr) | $\mu\, PMF$ | $\sigma\, PMF$ | $\mu\, S$ | $\sigma\, S$ |
|---|---|---|---|---|---|---|---|---|---|
| Manned | 3265277 | 3.851e+4 | 5.395e+8 | 5.240e+4 | 5.135e+8 | 6.501e-1 | 6.813e-3 | 9.084e-1 | 6.684e-5 |
| Unmanned | 3271293 | 3.779e+4 | 5.237e+8 | 3.506e+4 | 3.845e+8 | 6.579e-1 | 6.676e-3 | 9.099e-1 | 6.783e-5 |
| Manned&PV | 1633140 | 3.831e+4 | 5.382e+8 | 5.224e+4 | 5.135e+8 | 6.527e-1 | 6.622e-3 | 9.092e-1 | 6.639e-5 |
| Manned&RTG | 1632137 | 3.871e+4 | 5.407e+8 | 5.256e+4 | 5.134e+8 | 6.475e-1 | 6.990e-3 | 9.076e-1 | 6.610e-5 |
| Unmanned&PV | 1636035 | 3.760e+4 | 5.225e+8 | 3.523e+4 | 3.868e+8 | 6.606e-1 | 6.486e-3 | 9.107e-1 | 6.735e-5 |
| Unmanned&RTG | 1635258 | 3.799e+4 | 5.248e+8 | 3.489e+4 | 3.821e+8 | 6.553e-1 | 6.852e-3 | 9.092e-1 | 6.712e-5 |
| Manned&PV&20MLI | 816140 | 3.855e+4 | 5.436e+8 | 5.236e+4 | 5.195e+8 | 6.534e-1 | 6.597e-3 | 9.109e-1 | 6.333e-5 |
| Manned&PV&60MLI | 817000 | 3.806e+4 | 5.328e+8 | 5.212e+4 | 5.074e+8 | 6.520e-1 | 6.646e-3 | 9.074e-1 | 6.357e-5 |
| Manned&RTG&20MLI | 815536 | 3.902e+4 | 5.474e+8 | 5.275e+4 | 5.236e+8 | 6.478e-1 | 7.107e-3 | 9.093e-1 | 6.303e-5 |
| Manned&RTG&60MLI | 816601 | 3.840e+4 | 5.337e+8 | 5.236e+4 | 5.032e+8 | 6.472e-1 | 6.872e-3 | 9.059e-1 | 6.331e-5 |
| Unmanned&PV&20MLI | 817687 | 3.782e+4 | 5.265e+8 | 3.524e+4 | 3.869e+8 | 6.614e-1 | 6.472e-3 | 9.124e-1 | 6.413e-5 |
| Unmanned&PV&60MLI | 818348 | 3.738e+4 | 5.183e+8 | 3.522e+4 | 3.868e+8 | 6.598e-1 | 6.499e-3 | 9.090e-1 | 6.471e-5 |
| Unmanned&RTG&20MLI | 817197 | 3.827e+4 | 5.300e+8 | 3.489e+4 | 3.818e+8 | 6.556e-1 | 6.984e-3 | 9.109e-1 | 6.394e-5 |
| Unmanned&RTG&60MLI | 818061 | 3.772e+4 | 5.194e+8 | 3.489e+4 | 3.823e+8 | 6.549e-1 | 6.720e-3 | 9.075e-1 | 6.445e-5 |
| Manned&PV&20MLI&NE | 408083 | 3.852e+4 | 5.437e+8 | 5.226e+4 | 5.210e+8 | 6.540e-1 | 6.606e-3 | 9.116e-1 | 6.274e-5 |
| Manned&PV&20MLI&DS | 408057 | 3.859e+4 | 5.435e+8 | 5.247e+4 | 5.180e+8 | 6.529e-1 | 6.587e-3 | 9.101e-1 | 6.274e-5 |
| Manned&PV&60MLI&NE | 408516 | 3.803e+4 | 5.329e+8 | 5.201e+4 | 5.088e+8 | 6.526e-1 | 6.654e-3 | 9.082e-1 | 6.301e-5 |
| Manned&PV&60MLI&DS | 408484 | 3.810e+4 | 5.326e+8 | 5.222e+4 | 5.059e+8 | 6.514e-1 | 6.637e-3 | 9.067e-1 | 6.293e-5 |
| Manned&RTG&20MLI&NE | 407786 | 3.897e+4 | 5.476e+8 | 5.261e+4 | 5.253e+8 | 6.486e-1 | 7.119e-3 | 9.101e-1 | 6.251e-5 |
| Manned&RTG&20MLI&DS | 407750 | 3.907e+4 | 5.473e+8 | 5.289e+4 | 5.219e+8 | 6.470e-1 | 7.094e-3 | 9.086e-1 | 6.236e-5 |
| Manned&RTG&60MLI&NE | 408319 | 3.835e+4 | 5.339e+8 | 5.222e+4 | 5.048e+8 | 6.480e-1 | 6.881e-3 | 9.067e-1 | 6.271e-5 |
| Manned&RTG&60MLI&DS | 408282 | 3.845e+4 | 5.336e+8 | 5.250e+4 | 5.016e+8 | 6.464e-1 | 6.862e-3 | 9.051e-1 | 6.272e-5 |
| Unmanned&PV&20MLI&NE | 408858 | 3.778e+4 | 5.266e+8 | 3.525e+4 | 3.870e+8 | 6.620e-1 | 6.481e-3 | 9.132e-1 | 6.354e-5 |
| Unmanned&PV&20MLI&DS | 408829 | 3.785e+4 | 5.264e+8 | 3.523e+4 | 3.869e+8 | 6.609e-1 | 6.462e-3 | 9.117e-1 | 6.354e-5 |
| Unmanned&PV&60MLI&NE | 409189 | 3.735e+4 | 5.185e+8 | 3.523e+4 | 3.868e+8 | 6.604e-1 | 6.507e-3 | 9.098e-1 | 6.412e-5 |

(continued on next page)

(continued from previous page)

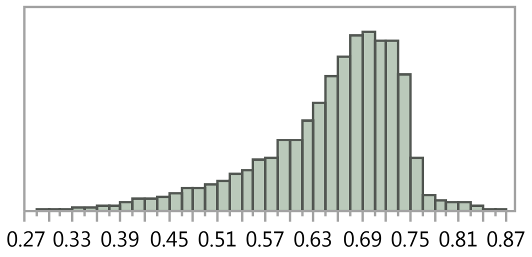
| **Description** | **N** | **$\mu\, m_{gross}$ (kg)** | **$\sigma\, m_{gross}$ (kg)** | **$\mu\, C_{gross}$ (MYr)** | **$\sigma\, C_{gross}$ (MYr)** | **$\mu\, PMF$** | **$\sigma\, PMF$** | **$\mu\, S$** | **$\sigma\, S$** |
|---|---|---|---|---|---|---|---|---|---|
| Unmanned&PV&60MLI&DS | 409159 | 3.741e+4 | 5.182e+8 | 3.522e+4 | 3.867e+8 | 6.592e-1 | 6.491e-3 | 9.082e-1 | 6.412e-5 |
| Unmanned&RTG&20MLI&NE | 408620 | 3.822e+4 | 5.301e+8 | 3.490e+4 | 3.819e+8 | 6.564e-1 | 6.995e-3 | 9.117e-1 | 6.337e-5 |
| Unmanned&RTG&20MLI&DS | 408577 | 3.832e+4 | 5.298e+8 | 3.488e+4 | 3.818e+8 | 6.548e-1 | 6.971e-3 | 9.101e-1 | 6.332e-5 |
| Unmanned&RTG&60MLI&NE | 409045 | 3.767e+4 | 5.196e+8 | 3.490e+4 | 3.824e+8 | 6.557e-1 | 6.729e-3 | 9.082e-1 | 6.386e-5 |
| Unmanned&RTG&60MLI&DS | 409016 | 3.776e+4 | 5.193e+8 | 3.488e+4 | 3.823e+8 | 6.541e-1 | 6.711e-3 | 9.067e-1 | 6.385e-5 |
| 1 Stage | 19109 | 4.172e+4 | 7.145e+8 | 2.184e+4 | 2.704e+8 | 7.285e-1 | 8.676e-3 | 9.826e-1 | 3.112e-5 |
| 2 Stage | 6517461 | 3.814e+4 | 5.311e+8 | 4.379e+4 | 5.234e+8 | 6.538e-1 | 6.738e-3 | 9.090e-1 | 5.220e-5 |
| 1 Stage Liquid | 14509 | 4.325e+4 | 7.673e+8 | 2.347e+4 | 2.643e+8 | 7.576e-1 | 1.903e-3 | 9.844e-1 | 2.349e-5 |
| 1 Stage Nuclear | 2296 | 4.167e+4 | 3.912e+8 | 2.709e+4 | 2.675e+8 | 5.057e-1 | 3.619e-3 | 9.769e-1 | 1.290e-5 |
| 1 Stage solid | 2304 | 3.208e+4 | 5.966e+8 | 6.329e+3 | 2.700e+7 | 7.669e-1 | 8.495e-5 | 9.769e-1 | 1.287e-5 |
| 2 Stage Liquid | 4026074 | 3.905e+4 | 5.780e+8 | 4.740e+4 | 5.335e+8 | 6.799e-1 | 2.717e-3 | 9.130e-1 | 2.386e-5 |
| 2 Stage Nuclear | 73728 | 3.966e+4 | 2.163e+8 | 4.769e+4 | 3.280e+8 | 4.172e-1 | 3.317e-3 | 8.943e-1 | 1.287e-5 |
| 2 Stage Solid | 73728 | 2.815e+4 | 4.212e+8 | 1.189e+4 | 4.917e+7 | 7.204e-1 | 1.532e-4 | 8.943e-1 | 1.287e-5 |
| 2 Stage Liquid-Nuclear | 589234 | 4.370e+4 | 3.347e+8 | 5.066e+4 | 4.719e+8 | 5.563e-1 | 3.056e-3 | 9.020e-1 | 2.318e-5 |
| 2 Stage Liquid-Solid | 589824 | 3.018e+4 | 4.455e+8 | 2.968e+4 | 2.759e+8 | 6.641e-1 | 2.592e-3 | 9.002e-1 | 2.317e-5 |
| 2 Stage Nuclear-Liquid | 512562 | 3.695e+4 | 4.327e+8 | 4.510e+4 | 4.097e+8 | 5.317e-1 | 4.993e-3 | 9.053e-1 | 1.350e-5 |
| 2 Stage Nuclear-Solid | 73728 | 2.863e+4 | 2.911e+8 | 2.697e+4 | 1.327e+8 | 4.898e-1 | 7.375e-3 | 8.943e-1 | 1.287e-5 |
| 2 Stage Solid-Liquid | 504855 | 3.709e+4 | 5.653e+8 | 3.021e+4 | 3.191e+8 | 7.294e-1 | 9.877e-4 | 9.053e-1 | 1.352e-5 |
| 2 Stage Solid-Nuclear | 73728 | 4.141e+4 | 3.221e+8 | 3.283e+4 | 2.203e+8 | 5.923e-1 | 1.922e-3 | 8.943e-1 | 1.287e-5 |
| 20MLI | 3266560 | 3.841e+4 | 5.371e+8 | 4.380e+4 | 5.294e+8 | 6.546e-1 | 6.814e-3 | 9.109e-1 | 6.481e-5 |
| 60MLI | 3270010 | 3.789e+4 | 5.262e+8 | 4.364e+4 | 5.187e+8 | 6.535e-1 | 6.705e-3 | 9.075e-1 | 6.521e-5 |
| Small Payload | 3354889 | 1.889e+4 | 1.741e+8 | 4.058e+4 | 4.692e+8 | 6.457e-1 | 8.113e-3 | 9.100e-1 | 6.702e-5 |
| Large Payload | 3181681 | 5.846e+4 | 1.048e+8 | 4.703e+4 | 5.605e+8 | 6.629e-1 | 5.181e-3 | 9.083e-1 | 6.742e-5 |
| PV | 3269175 | 3.795e+4 | 5.305e+8 | 4.373e+4 | 5.224e+8 | 6.567e-1 | 6.570e-3 | 9.099e-1 | 6.748e-5 |
| RTG | 3267395 | 3.835e+4 | 5.328e+8 | 4.371e+4 | 5.257e+8 | 6.514e-1 | 6.936e-3 | 9.084e-1 | 6.722e-5 |

(continued on next page)

(continued from previous page)

| **Description** | **N** | $\mu\, m_{gross}$ **(kg)** | $\sigma\, m_{gross}$ **(kg)** | $\mu\, C_{gross}$ **(MYr)** | $\sigma\, C_{gross}$ **(MYr)** | $\mu\, PMF$ | $\sigma\, PMF$ | $\mu\ S$ | $\sigma\ S$ |
|---|---|---|---|---|---|---|---|---|---|
| Disk | 2600408 | 4.025e+4 | 5.746e+8 | 4.702e+4 | 5.804e+8 | 6.540e-1 | 4.594e-3 | 9.091e-1 | 5.331e-5 |
| Single | 1316929 | 3.645e+4 | 4.664e+8 | 3.588e+4 | 4.153e+8 | 6.131e-1 | 1.454e-2 | 9.031e-1 | 5.348e-5 |
| Stacked | 2619233 | 3.693e+4 | 5.146e+8 | 4.439e+4 | 4.807e+8 | 6.747e-1 | 3.730e-3 | 9.123e-1 | 6.176e-5 |
| Manned&Disk | 1297806 | 4.093e+4 | 5.880e+8 | 5.747e+4 | 5.542e+8 | 6.469e-1 | 4.777e-3 | 9.083e-1 | 5.147e-5 |
| Manned&Single | 658463 | 3.645e+4 | 4.665e+8 | 4.353e+4 | 4.168e+8 | 6.131e-1 | 1.453e-2 | 9.023e-1 | 5.288e-5 |
| Manned&Stacked | 1309008 | 3.714e+4 | 5.183e+8 | 5.183e+4 | 4.563e+8 | 6.720e-1 | 3.769e-3 | 9.115e-1 | 6.111e-5 |
| Unmanned&Disk | 1302602 | 3.956e+4 | 5.602e+8 | 3.662e+4 | 3.894e+8 | 6.610e-1 | 4.311e-3 | 9.099e-1 | 5.389e-5 |
| Unmanned&Single | 658466 | 3.645e+4 | 4.664e+8 | 2.823e+4 | 2.965e+8 | 6.131e-1 | 1.454e-2 | 9.039e-1 | 5.288e-5 |
| Unmanned&Stacked | 1310225 | 3.671e+4 | 5.109e+8 | 3.695e+4 | 3.943e+8 | 6.774e-1 | 3.676e-3 | 9.130e-1 | 6.121e-5 |

**Vehicle PMF**

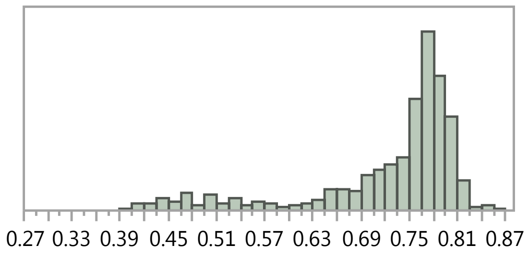


| Quantiles | |
|---|---|
| maximum | 0.8659680349 |
| minimum | 0.2938305038 |

| Summary Statistics | |
|---|---|
| Mean | 0.6540273 |
| Std Dev | 0.082217 |
| Std Err Mean | 3.2158e-5 |
| Upper 95% Mean | 0.6540903 |
| Lower 95% Mean | 0.6539643 |
| N | 6536570 |

**Figure 102:** Experiment 3 Total Vehicle PMF Distribution

**Vehicle PMF**

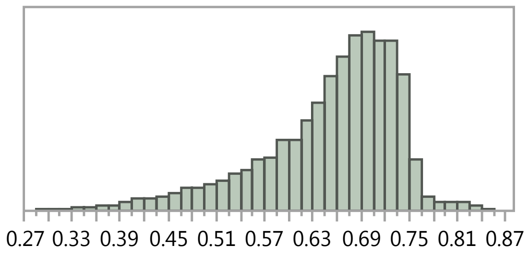


| Quantiles | |
|---|---|
| maximum | 0.3678466147 |
| minimum | 0.2938305038 |

| Summary Statistics | |
|---|---|
| Mean | 0.7284669 |
| Std Dev | 0.0931467 |
| Std Err Mean | 0.0006738 |
| Upper 95% Mean | 0.7297877 |
| Lower 95% Mean | 0.7271462 |
| N | 19109 |

**Figure 103:** Total Vehicle PMF Distribution of Architectures with 1 Stage

**Vehicle PMF**



| Quantiles | |
|---|---|
| maximum | 0.8498267227 |
| minimum | 0.2938305038 |

| Summary Statistics | |
|---|---|
| Mean | 0.6538091 |
| Std Dev | 0.0820836 |
| Std Err Mean | 3.2153e-5 |
| Upper 95% Mean | 0.6538721 |
| Lower 95% Mean | 0.653746 |
| N | 6517461 |

**Figure 104:** Total Vehicle PMF Distribution of Architectures with 2 Stages

# APPENDIX H

# MODEL VALIDATION RESULTS

The following tables contain validation data for the models developed through this body of work. Table 61 shows the calculated mass of a notional vehicle throughout a set of mission events. The difference is between the calculation when utilizing DYREQT and the mission event models developed for this dissertation compared to an industry-developed tool of similar fidelity, HExAM. Table 62 provides the sizing calculations for 10 propulsive stages, ranging from designs built and flown to conceptual designs. Sizing utilized the set of subsystem models developed and discussed in Chapter 4.5.2 of this dissertation. Differences were recorded for both the inert mass and the propellant mass fraction.

**Table 61:** Mission Model Validation Data

| | | Initial Mass (kg) | | | Final Mass (kg) | | |
|---|---|---|---|---|---|---|---|
| **MET(d)** | **Event** | **HExAM** | **Estimated** | **% Diff.** | **HExAM** | **Estimated** | **% Diff.** |
| 2 | Launch Clean Up | 3071 | 3071.0 | 0.00 | 3019 | 3019.2 | 0.01 |
| 5 | Powered LGA | 3019 | 3019.2 | 0.01 | 2817 | 2816.8 | -0.01 |
| 7 | Target DRO | 2817 | 2816.8 | -0.01 | 2803 | 2802.5 | -0.02 |
| 9 | DRO Insertion | 2803 | 2802.5 | -0.02 | 2615 | 2614.7 | -0.01 |
| 12 | Clean Up | 2615 | 2614.7 | -0.01 | 2610 | 2610.2 | 0.01 |
| 12 | Drop Adapter | 2610 | 2610.2 | 0.01 | 2460 | 2460.2 | 0.01 |
| 12 | Drop Payload | 2460 | 2460.2 | 0.01 | 1460 | 1460.2 | 0.01 |
| 12 | Disposal | 1460 | 1460.2 | 0.01 | 1450 | 1450.3 | 0.02 |
| 12 | Drop Stage | 1450 | 1450.3 | 0.02 | 0 | 0.0 | 0.00 |
| 12 | Connect Payload | 0 | 0.0 | 0.00 | 1000 | 1000.0 | 0.00 |

**Table 62:** Vehicle Subsystem Model Validation Data

| | | | **Inert Mass (kg)** | | | **PMF** | | |
|---|---|---|---|---|---|---|---|---|
| **Element** | **Ref.** | **Descriptions** | **Lit.** | **Est.** | **% Diff.** | **Lit.** | **Est.** | **% Diff.** |
| S-IVB | [119] | large LOX/$LH_2$ cryogenic stage | 12900 | 13040.7 | 1.09 | 0.90 | 0.90 | 0.00 |
| SEP | [89] | large Xenon solar-electric stage | 10042 | 9567.9 | -4.72 | 0.59 | 0.61 | 3.39 |
| MCPS | [90] | large active cooled LOX/$LCH_4$ cryogenic stage | 10514.8 | 9305.6 | -11.50 | 0.82 | 0.84 | 2.44 |
| NTP | [81] | cermet core nuclear thermal propulsion stage | 33450 | 32145.1 | -3.90 | 0.65 | 0.67 | 3.08 |
| Centaur[1] | [53] | small passive cooled LOX/$LH_2$ cryogenic stage | 2247 | 2790.8 | 24.20 | 0.92 | 0.88 | -4.35 |
| Centaur[2] | [53] | small passive cooled LOX/$LH_2$ cryogenic stage | 2462 | 3079.1 | 25.06 | 0.91 | 0.87 | -4.40 |
| SSPS[3] | [53] | small $N_2/O_4$/Aerozine 50 storable upper stage | 950 | 997.6 | 5.01 | 0.86 | 0.86 | 0.00 |
| SSPS[4] | [53] | small passive cooled LOX/$LH_2$ cryogenic stage | 2850 | 2761.9 | -3.09 | 0.88 | 0.88 | 0.00 |
| SSPS[5] | [53] | small passive cooled LOX/$LH_2$ cryogenic stage | 3490 | 3402.2 | -2.52 | 0.89 | 0.89 | 0.00 |
| EPS-5E | [53] | small $N_2/O_4$/MMH storable upper stage | 1200 | 1229.8 | 2.48 | 0.89 | 0.89 | 0.00 |

[1]single engine
[2]twin engine
[3]Delta II upper stage
[4]Delta IV-M upper stage
[5]Delta IV-H upper stage

# APPENDIX I

# MODEL SOURCE CODES

This appendix provides the source code for the models developed for use throughout this dissertation. This includes: vehicle subsystem models, mission event models, fluids models, and the TransCost model, utilized in both the experiments and proof of concept. Descriptions of each model, its inputs/outputs, and references can be found throughout the source code. The source code is written in Python 3. The classes defined in these models inherit from DYREQT, which in turn inherits much of its structure and functionality from the OpenMDAO module, with the exception of the cost model. DYREQT source code is not provided in this dissertation.

## *I.1 Avionics SubElement Model*

```python
# -*- coding: utf-8 -*-
"""
Description:
    A DYREQT Avionics subelement for Douglas Trent's PhD

Written by:
    Douglas J. Trent
    NASA Marshall Space Flight Center
    Advanced Concept Office
    douglas.trent@nasa.gov

Created: 04/06/2017
Revised: 04/19/2017
"""
# import DYREQT Subelement base class
from SubElements import SubElement
from numpy import zeros

# create the structures subelement
class AvionicsPhD(SubElement):
    """Estimates the mass of a power subsystem. Much of the mass of the
    components are derived from Space Mission Engineering: The new SMAD by
    James R. Wertz et. al.

    Input Params
    ------------
    actuators : list
        The type of attitude control device(s). For multiples of the same
        sensor, include the number multiple times.

            0 = Reaction Wheels
        1 = Control Moment Gyros

        2 = Magnetic Torquers

sensors : list
     A list of predefined avionics sensor(s). For multiples of the same
     sensor, include the number multiple times.

         0 = Gyros

         1 = Sun Sensor

         2 = Star Sensor (scanner)

         3 = Star Sensor (fixed)

         4 = Horizon Sensor

         5 = Magnetometer


comms_type : int
    The type of communications package.

        0 = None

        1 = Near Earth

        2 = Deep Space

accuracy : float
    A factor to determine sensor accuracy for scaling mass. A value ranging
    from 0.0 to 1.0. Higher values correspond to higher accuracy, resulting
    in increased mass and power requirements.

wireless_sensors : bool(False)
    If True, assumes wireless transmission of data without the need for
    cables, reducing cable mass.

additional_devices : list
    A list of other additional fixed devices to include in the system. Each
    item in the list is a list which defines a the mass(kg) and
    power required(W) of the device, in the form [m,P]

Inherited Params
----------------
dry_mass : array
    The dry mass of the element (kg) for scaling actuators

Outputs
-------
inert_mass : float
    The inert mass of the subelement (kg)

heat_loads : array
    The amount of heat generated by other subsystems to be dissipated by
    the radiators (W)

power_req : array
    The power required from each element subsystems (W)
"""

def __init__(self,**kwargs):
    super().__init__(**kwargs)
    # user model inputs
    self.add_param(self.base_name+'_actuators', val=list())
    self.add_param(self.base_name+'_sensors', val=list())
    self.add_param(self.base_name+'_comms_type', val=int(0))
```

```
        self.add_param(self.base_name+'_accuracy', val=float(1))
        self.add_param(self.base_name+'_wireless_sensors', val=False)
        self.add_param(self.base_name+'_additional_devices', val=list())
        # parameters from the element (DYREQT internal or from other subelements)
        self.add_param(self.element.base_name+'_dry_mass', val=zeros(self.element.num_events+1),
        ↪  units='kg')
        # outputs used by the parent element or subelements
        self.add_output(self.base_name+'_heat_load', val=float(0), units='W')
        self.add_output(self.base_name+'_power_req', val=float(0), units='W')
        self.add_output(self.base_name+'_inert_mass', val=float(0), units='kg')

    def pre_setup(self,problem):
        pass

    def post_setup(self, problem):
        """Check inputs after final connections have been made
        """

        actuators = self.params[self.base_name+'_actuators']
        sensors = self.params[self.base_name+'_sensors']
        comms_type = self.params[self.base_name+'_comms_type']
        accuracy = self.params[self.base_name+'_accuracy']
        add_devices = self.params[self.base_name+'_additional_devices']

        if accuracy < 0. or accuracy > 1.:
            msg = ('accuracy must be a value between 0.0 and 1.0')
            raise Exception(msg)

        for val in actuators:
            if val not in [0,1,2]:
                msg = ('invalid control device. values must be in the range [0:2]')
                raise Exception(msg)

        for val in sensors:
            if val not in [0,1,2,3,4,5]:
                msg = ('invalid sensor device. values must be in the range [0:5]')
                raise Exception(msg)

        if comms_type not in [0,1,2]:
            msg = ('invalid comms_type selection. must be either 0 for None, '
                   '1 for near earth, or 2 for deep space')
            raise Exception(msg)

        for idx,device in enumerate(add_devices):
            if len(device) != 2:
                msg = ('additional devices must define a mass and power property')
                raise Exception(msg)
            for val in device:
                if type(val) != float and type(val) != int:
                    msg = ('invalid additional device {0}, mass and power '
                           'definitions must be of type float or int'.format(idx))
                    raise Exception(msg)

    def solve_nonlinear(self, params, unknowns, resids):

        actuators = self.params[self.base_name+'_actuators']
        sensors = self.params[self.base_name+'_sensors']
        comms_type = self.params[self.base_name+'_comms_type']
        accuracy = self.params[self.base_name+'_accuracy']
        wireless_sensors = self.params[self.base_name+'_wireless_sensors']
        add_devices = self.params[self.base_name+'_additional_devices']
        element_mass = max(params[self.element.base_name+'_dry_mass'])

        actuator_mass = 0.
        actuator_pwr = 0.
        for val in actuators:
            m_range = [0.,0.]; P_range = [0.,0.]
            if val == 0:
```

```
            m_range = [2.,20.]; P_range = [10.,100.]
        elif val == 1:
            m_range = [0.1,10.]; P_range = [90.,150.]
        elif val == 2:
            m_range = [0.4,50.]; P_range = [0.6,16.]

        if element_mass < 10000.:
            comp_mass = ((m_range[1]-m_range[0])/10000.)*element_mass + m_range[0]
            comp_pwr = ((P_range[1]-P_range[0])/10000.)*element_mass + P_range[0]
        else:
            comp_mass = m_range[1]
            comp_pwr = P_range[1]
        actuator_mass += comp_mass
        actuator_pwr += comp_pwr

    sensor_mass = 0.
    sensor_pwr = 0.
    for val in sensors:
        m_range = [0.,0.]; P_range = [0.,0.]
        if val == 0:
            m_range = [0.1,15.]; P_range = [0.6,16.]
        elif val == 1:
            m_range = [0.1,2.]; P_range = [0.,3.]
        elif val == 2:
            m_range = [2.,5.]; P_range = [0.6,16.]
        elif val == 3:
            m_range = [1.,4.]; P_range = [5.,10.]
        elif val == 4:
            m_range = [0.5,3.5]; P_range = [0.3,5.]
        elif val == 5:
            m_range = [0.3,1.2]; P_range = [0.0,1.]

        comp_mass = ((m_range[1]-m_range[0]))*accuracy + m_range[0]
        comp_pwr = ((P_range[1]-P_range[0]))*accuracy + P_range[0]
        sensor_mass += comp_mass
        sensor_pwr += comp_pwr

    comms_mass = 0.
    comms_pwr = 0.
    if comms_type == 0:
        pass
    elif comms_type == 1:
        comms_mass = 20.7
        comms_pwr = 104.0
    elif comms_type == 2:
        comms_mass = 42.0
        comms_pwr = 165.0

    add_devices_mass = 0.
    add_devices_pwr = 0.
    for device in add_devices:
        add_devices_mass += device[0]
        add_devices_pwr += device[1]

    # accounts for 4% cable mass
    cable_factor = 0.96
    if wireless_sensors:
        cable_factor = 0.99
    inert_mass = (actuator_mass + sensor_mass + comms_mass + add_devices_mass) / cable_factor
    total_power = actuator_pwr + sensor_pwr + comms_pwr + add_devices_pwr
    heat_load = 0.9*total_power

    # assign unknowns
    unknowns[self.base_name+'_heat_load'] = heat_load
    unknowns[self.base_name+'_power_req'] = total_power
    unknowns[self.base_name+'_inert_mass'] = inert_mass
```

## I.2 Engine SubElement Model

```
# -*- coding: utf-8 -*-
"""
Description:
    A DYREQT Engine subelement for Douglas Trent's PhD

Written by:
    Douglas J. Trent
    NASA Marshall Space Flight Center
    Advanced Concept Office
    douglas.trent@nasa.gov

Created: 04/06/2017
Revised: 07/31/2017
"""
# import DYREQT Subelement base class
from SubElements import SubElement
from Constants import G0
# import other modules
import FluidsDef as fluids
from numpy import zeros, floor, ceil, pi, log, log10, exp, sqrt, tan
from scipy.optimize import brentq

# create the structures subelement
class EnginesPhD(SubElement):
    """Liquid engine sizer from equations in Space Propulsion Analysis and
    Design, by Ronald W. Humble, et. al. Sec. 5.3.1

    Input Params
    ------------
    mps_class : str('liquid')
        The class of the main propulsion system.
        One of ['liquid','solid','nuclear','electric']

    propellants_mps : str('lox/lh2')
        The oxidizer and Fuel of the propulsion system, separated by a forward
        slash (/). For mono-propellant, only specify a fuel with no slash (/).
        This input is ignored when mps_class is set to 'solid'.

    total_thrust_mps : float(100 kN)
        The total thrust of the main propulsion system (kN)

    isp_mps : float(350 s)
        The specific impulse of the main propulsion system (s)

    mixture_ratio_mps : float(1) (optional, required if mps_class is 'liquid')
        The mass ratio of the oxidizer to fuel of the main propulsion system.
        This will be ignored if a mono-propellant is specified in the
        propellants_mps input.

    start_penalty_mps : float(0.0 kg)
        A mass of propellant lost during engine startup of the main propulsion
        system (kg)

    engine_thrust_mps : float(25 kN)
        The thrust per engine of the main propulsion system (kN)

    core_type : str('CERMET') (optional, required if mps_class is 'nuclear')
        The type of nuclear core, one of ['PBR','CERMET']

    T_chamber : str(2800 K) (optional, required if mps_class is 'nuclear')
        The maximum chamber temperature of the engine which the propellant will
        be heated to (K)

    P_chamber : str(3.5 MPa) (optional, required if mps_class is 'nuclear')
        The pressure of the propellant in the reactor chamber (MPa)
```

```
thruster_efficiency_mps = float(0.5) (optional, required if mps_class is 'electric')
    The electric thruster efficiency of the main propulsion system. Ideal
    thrusters have an efficiency of 1.

thruster_specific_mass_mps = float(7 kg/kW) (optional, required if mps_class is 'electric')
    The electric thruster mass per unit power (kg/kW) of the main propulsion
    system.

thruster_power_mps = float(1 kW) (optional, required if mps_class is 'electric')
    The input power required per thruster of the main propulsion system (kW)

redundancy_mps = float(0.2) (optional, required if mps_class is 'electric')
    The redundancy of electric thrusters in the main propulsion system.
    A value of 1 corresponds to 100% redundancy.

power_mgmt_specific_mass_mps = float(6 kg/kW) (optional, required if mps_class is 'electric')
    The mass per unit power of the power management system for the main
    propulsion system (kg/kW)

rcs_class : str('liquid')
    The class of the reaction control system.
    One of ['liquid','electric']

propellants_rcs : str('hydrazine')
    The oxidizer and Fuel of the propulsion system, separated by a forward
    slash (/). For mono-propellant, only specify a fuel with no slash (/).

total_thrust_rcs : float(1 kN)
    The total thrust of the reaction control system (kN)

isp_rcs : float(300 s)
    The specific impulse of the reaction control system (s)

mixture_ratio_rcs : float(1) (optional, required if rcs_class is 'liquid')
    The mass ratio of the oxidizer to fuel of the reaction control system.
    This will be ignored if a mono-propellant is specified in the
    propellants_rcs input.

start_penalty_rcs : float(0.0 kg)
    A mass of propellant lost during engine startup of the reaction control
    system (kg)

engine_thrust_rcs : float(0.25 kN)
    The thrust per engine of the reaction control system (kN)

thruster_efficiency_rcs = float(0.5) (optional, required if rcs_class is 'electric')
    The electric thruster efficiency of the reaction control system. Ideal
    thrusters have an efficiency of 1.

thruster_specific_mass_rcs = float(7 kg/kW) (optional, required if rcs_class is 'electric')
    The electric thruster mass per unit power (kg/kW) of the reaction
    control system.

thruster_power_rcs = float(1 kW) (optional, required if rcs_class is 'electric')
    The input power required per thruster of the reaction control system(kW)

redundancy_rcs = float(0.2) (optional, required if rcs_class is 'electric')
    The redundancy of electric thrusters in the reaction control system.
    A value of 1 corresponds to 100% redundancy.

power_mgmt_specific_mass_rcs = float(6 kg/kW) (optional, required if rcs_class is 'electric')
    The mass per unit power of the power management system for the reaction
    control system (kg/kW)

Inherited Params
----------------
max_propellant_mass_mps : float
    The total amount of propellant required for all impulsive maneuvers
```

```
        performed by the main propulsion system (kg)

    Outputs
    -------
    propellants_mps : str
        The oxidizer and Fuel of the propulsion system, separated by a forward
        slash (/). For mono-propellant, only specify a fuel with no slash (/).

    isp_mps : float
        The specific impulse of the main propulsion system (s)

    thrust_mps : float
        The total thrust of the main propulsion system (kN)

    mixture_ratio_mps : float
        The mixture ratio of the main propulsion system propellants

    mps_start_penalty : float
        A mass of propellant lost during engine startup of the main propulsion
        system (kg)

    propellants_rcs : str
        The oxidizer and Fuel of the propulsion system, separated by a forward
        slash (/). For mono-propellant, only specify a fuel with no slash (/).

    isp_rcs : float
        The specific impulse of the reaction control system (s)

    thrust_rcs : float
        The total thrust of the reaction control system (kN)

    mixture_ratio_rcs : float
        The mixture ratio of the reaction control system propellants

    rcs_start_penalty : float
        A mass of propellant lost during engine startup of the reaction control
        system (kg)

    power_req : array
        The power required for the thermal subelement (W)

    inert_mass : float
        The inert mass of the subelement (kg)
    """

    def __init__(self,**kwargs):
        super().__init__(**kwargs)
        # user model inputs
        self.add_param(self.base_name+'_mps_class', val=str('liquid'))
        self.add_param(self.base_name+'_rcs_class', val=str('liquid'))
        self.add_param(self.base_name+'_propellants_mps', val=str('lox/lh2'))
        self.add_param(self.base_name+'_propellants_rcs', val=str('hydrazine'))
        self.add_param(self.base_name+'_total_thrust_mps', val=float(100), units='kN')
        self.add_param(self.base_name+'_total_thrust_rcs', val=float(1), units='kN')
        self.add_param(self.base_name+'_isp_mps', val=float(350), units='s')
        self.add_param(self.base_name+'_isp_rcs', val=float(300), units='s')
        self.add_param(self.base_name+'_mixture_ratio_mps', val=float(1))
        self.add_param(self.base_name+'_mixture_ratio_rcs', val=float(1))
        self.add_param(self.base_name+'_start_penalty_mps', val=float(0), units='kg')
        self.add_param(self.base_name+'_start_penalty_rcs', val=float(0), units='kg')
        self.add_param(self.base_name+'_engine_thrust_mps', val=float(25.), units='kN')
        self.add_param(self.base_name+'_engine_thrust_rcs', val=float(0.25), units='kN')
        self.add_param(self.base_name+'_thruster_efficiency_mps', val=float(0.5))
        self.add_param(self.base_name+'_thruster_efficiency_rcs', val=float(0.5))
        self.add_param(self.base_name+'_thruster_specific_mass_mps', val=float(7), units='kg/kW')
        self.add_param(self.base_name+'_thruster_specific_mass_rcs', val=float(7), units='kg/kW')
        self.add_param(self.base_name+'_thruster_power_mps', val=float(1), units='kW')
        self.add_param(self.base_name+'_thruster_power_rcs', val=float(1), units='kW')
        self.add_param(self.base_name+'_redundancy_mps', val=float(0.2))
        self.add_param(self.base_name+'_redundancy_rcs', val=float(0.2))
        self.add_param(self.base_name+'_power_mgmt_specific_mass_mps', val=float(6), units='kg/kW')
        self.add_param(self.base_name+'_power_mgmt_specific_mass_rcs', val=float(6), units='kg/kW')
        self.add_param(self.base_name+'_core_type', val=str('cermet'))
        self.add_param(self.base_name+'_T_chamber', val=float(2800.), units='K')
        self.add_param(self.base_name+'_P_chamber', val=float(3.5), units='MPa')
        # parameters from the element (DYREQT internal or from other subelements)
        self.add_param(self.element.base_name+'_max_propellant_mass_mps', val=float(0), units='kg')
        # outputs inherited by the element (for use by other subelements or DYREQT)
        self.add_output(self.element.base_name+'_propellants_mps', val=str('lox/lh2'))
        self.add_output(self.element.base_name+'_propellants_rcs', val=str('lox/lh2'))
        self.add_output(self.element.base_name+'_thrust_mps', val=float(1), units='kN')
        self.add_output(self.element.base_name+'_thrust_rcs', val=float(1), units='kN')
        self.add_output(self.element.base_name+'_isp_mps', val=float(1), units='s')
        self.add_output(self.element.base_name+'_isp_rcs', val=float(1), units='s')
        self.add_output(self.element.base_name+'_mixture_ratio_mps', val=float(1))
        self.add_output(self.element.base_name+'_mixture_ratio_rcs', val=float(1))
        self.add_output(self.element.base_name+'_mps_start_penalty', val=float(0), units='kg')
        self.add_output(self.element.base_name+'_rcs_start_penalty', val=float(0), units='kg')
        # outputs used by the parent element of subelement
        self.add_output(self.base_name+'_power_req', val=float(0), units='W')
        self.add_output(self.base_name+'_inert_mass', val=float(0), units='kg')

    def pre_setup(self,problem):
        pass

    def post_setup(self, problem):
        """Check inputs after final connections have been made
        """

        for propsys in ['mps','rcs']:

            engine_class = self.params[self.base_name+'_'+propsys+'_class'].lower()
            if not engine_class:
                msg = ('must specify a system class for the {0}'.format(propsys))
                raise Exception(msg)
            elif engine_class not in ['liquid','solid','electric','nuclear','massless']:
                msg = ('{0}_class {1} is an undefined class'.format(propsys,engine_class))
                raise Exception(msg)

            if engine_class != 'solid':
                propellants = self.params[self.base_name+'_propellants_'+propsys].lower()
                if not propellants:
                    msg = ('must specify propellants_{0} with system class "{1}"'.format(
                            propsys,engine_class))
                    raise Exception(msg)

            if propsys == 'rcs' and engine_class not in ['liquid','electric']:
                msg = ("'{0}' is an invalid class for the RCS, choose one of "
                       "['liquid','electric']".format(engine_class))
                raise Exception(msg)

            if engine_class == 'electric':
                thruster_efficiency = self.params[self.base_name+'_thruster_efficiency_'+propsys]
                redundancy = self.params[self.base_name+'_redundancy_'+propsys]
                if thruster_efficiency > 1:
                    msg = ('thruster_efficiency_{0} must be a value between '
                           'zero and one'.format(propsys))
                    raise Exception(msg)
                if redundancy < 0:
                    msg = ('redundancy_{0} must be non-negative'.format(propsys))
                    raise Exception(msg)

            # the nuclear engine model performs its own input checks

    def solve_nonlinear(self, params, unknowns, resids):
        inert_mass = 0.
        power_req = 0.

        for propsys in ['mps','rcs']:

            engine_class = params[self.base_name+'_'+propsys+'_class'].lower()
            propellants  = params[self.base_name+'_propellants_'+propsys].lower()
            total_thrust = params[self.base_name+'_total_thrust_'+propsys]
            isp = params[self.base_name+'_isp_'+propsys]
            mixture_ratio = params[self.base_name+'_mixture_ratio_'+propsys]
            start_penalty = params[self.base_name+'_start_penalty_'+propsys]

            # size the engine subelement for the prop system
            mass = 0.
            power = 0.

            if engine_class == 'liquid':
                engine_thrust = params[self.base_name+'_engine_thrust_'+propsys]
                mass,power = _liquid(total_thrust, engine_thrust, propellants)
            elif engine_class == 'solid':
                prop_load = params[self.element.base_name+'_max_propellant_mass_mps']
                mass,power = _solid(prop_load)
            elif engine_class == 'electric':
                thruster_efficiency = params[self.base_name+'_thruster_efficiency_'+propsys]
                thruster_specific_mass = params[self.base_name+'_thruster_specific_mass_'+propsys]
                thruster_power = params[self.base_name+'_thruster_power_'+propsys]
                redundancy = params[self.base_name+'_redundancy_'+propsys]
                power_mgmt_specific_mass = params[self.base_name+'_power_mgmt_specific_mass_'+propsys]
                mass,power = _electric(isp, total_thrust, thruster_efficiency,
                                       thruster_specific_mass, thruster_power,
                                       redundancy,power_mgmt_specific_mass)
            elif engine_class == 'nuclear':
                propellant = propellants
                engine_thrust = params[self.base_name+'_engine_thrust_'+propsys]
                core_type = params[self.base_name+'_core_type'].lower()
                T_chamber = params[self.base_name+'_T_chamber']
                P_chamber = params[self.base_name+'_P_chamber']
                mass,power = _nuclear(isp, total_thrust, engine_thrust, propellant, core_type,
                                      T_chamber,P_chamber)

            # adjust engine mass for RCS to include 3 additional off axis thrusters
            # for a total of 4 thrusters per thrust pod
            if propsys == 'rcs':
                mass = 4*mass

            inert_mass += mass
            power_req += power

            # assign system unknowns
            unknowns[self.element.base_name+'_propellants_'+propsys] = propellants
            unknowns[self.element.base_name+'_thrust_'+propsys] = total_thrust
            unknowns[self.element.base_name+'_isp_'+propsys] = isp
            unknowns[self.element.base_name+'_mixture_ratio_'+propsys] = mixture_ratio
            unknowns[self.element.base_name+'_'+propsys+'_start_penalty'] = start_penalty

        # assign unknowns
        unknowns[self.base_name+'_power_req'] = power_req
        unknowns[self.base_name+'_inert_mass'] = inert_mass

def _liquid(total_thrust, engine_thrust, propellants):
    """Liquid engine sizer from equations in Space Propulsion Analysis and
    Design, by Ronald W. Humble, et. al. Sec. 5.3.1

    Args
    ----
    total_thrust : float
        Total thrust of the propulsion system (kN)
    engine_thrust : float
        The thrust per engine (kN)

    propellants : str
        The propellants of the propulsion system, separated by a forward slash (/)

    Returns
    -------
    inert_mass : float
        The inert mass of the the engine system (kg)

    power_req : float
        The required power (W)
    """

    inert_mass = 0.
    power_req = 0.
    G0.convert_to_unit('m/s**2')

    num_engines = ceil(total_thrust / engine_thrust)
    thrust = engine_thrust*1000

    if propellants.count('/') == 0: # mono-propellant
        engine_mass = thrust / G0.value / (-3.7405e-10*thrust**4 + 7.1685e-7*thrust**3 +
                                     -5.2221e-4*thrust**2 + 0.18761*thrust +
                                     -0.039763)
    elif propellants.count('/') == 1: # bi-props
        if thrust < 50000:
            engine_mass = thrust / G0.value / (6.098e-4*thrust + 13.44)
        else:
            engine_mass = thrust / G0.value / (25.2*log10(thrust) - 80.7)
    else:
        raise Exception('may only specify up to two propellants')

    engine_mass = engine_mass * num_engines

    prop_mgmt_mass = 0.1 * engine_mass # valves, regulators, filters, transducers, etc.
    miscellaneous_hardware = 0.15 * (engine_mass + prop_mgmt_mass) # plumbing, brackets, insulation,

    inert_mass = engine_mass + prop_mgmt_mass + miscellaneous_hardware

    return inert_mass, power_req

def _solid(prop_load):
    """Solid motor sizer based on data points from Space Propulsion Analysis
    and Design, by Ronald W. Humble, et. al. Sec. 6.3. This model is a spline
    of two linear curve fits of the data. This is due to larger motors
    being built with joints and large thrust vector control mechanisms which
    drive the mass fractions of these larger motors back down with increasing
    prop loads.

    Args
    ----
    prop_load : float
        The propellant load of the solid motor, in kg.

    Returns
    -------
    inert_mass : float
        The inert mass of the the engine system (kg)

    power_req : float
        The required power (W)
    """

    inert_mass = 0.
    power_req = 0.
    if prop_load < 10000:
        if prop_load <= 0:
            f_prop = 1.
        else:
            f_prop = 0.0181*log(prop_load) + 0.7962
    else:
        f_prop = 0.95*exp(-1e-7*prop_load)

    inert_mass = (prop_load / f_prop) - prop_load

    return inert_mass, power_req

def _electric(isp, total_thrust, thruster_efficiency,
              thruster_specific_mass, thruster_power, redundancy,
              power_mgmt_specific_mass):
    """
    All empirical parameters are inputs
    Calculations are purely physics-based, so no applicability limits
    Source; Level0 EP tool from Dan Thomas (NASA Marshall Space Flight Center,
    Advanced Concepts Office), with some adaptations.

    Args
    ----
    isp : float
        specific impulse (sec)

    total_thrust : float
        Total thrust of the propulsion system (kN)

    thruster_efficiency : float
        thruster efficiency (dimensionless)

    thruster_specific_mass : float
        thruster specific mass (mass/input power) (kg/kW)

    thruster_power : float
        input power required for one thruster (kW)

    redundancy : int
        fraction of total number of thrusters

    power_mgmt_specific_mass : float
        power management system specific mass (kg/kW)

    Returns
    -------
    inert_mass : float
        total mass of the thruster system (kg)

    power_req : float
        total power required by the thruster system (W)
    """

    inert_mass = 0.
    power_req = 0.
    G0.convert_to_unit('m/s**2')

    #calculation of single-thruster parameters
    exhaust_velocity = G0.value*isp # m/s
    thruster_mass = thruster_specific_mass * thruster_power # kg/thruster
    jet_power_per_thruster = thruster_efficiency * thruster_power # kW
    thrust_per_thruster = 1000 * 2 * jet_power_per_thruster / exhaust_velocity # N

    #calculation of system parameters
    num_operating_thrusters = ceil(total_thrust * 1000. / thrust_per_thruster)
    power_req = num_operating_thrusters * thruster_power * 1000.
    num_total_thrusters = ceil(num_operating_thrusters * (1 + redundancy))
    engine_mass = thruster_mass * num_total_thrusters # kg
    # power management sizer
    # default value from Space Mission Engineering: The New SMAD,
    # sec. 18.6.3.2, p.550
    power_mgmt_mass = power_mgmt_specific_mass * power_req / 1000.

    # range from 5-10 kg as a function of total number of thrusters,
    # adapted from mass range estimate from Space Mission Engineering: The New
    # SMAD, sec. 18.6.3.2, p.550
    prop_mgmt_mass = 5 * (1 + (1 - (1/num_total_thrusters)))
    miscellaneous_hardware = 0.15 * (engine_mass + prop_mgmt_mass) # plumbing, brackets, insulation,

    inert_mass = engine_mass + prop_mgmt_mass + power_mgmt_mass + miscellaneous_hardware

    return inert_mass, power_req

def _nuclear(isp, total_thrust, engine_thrust, propellant, core_type, T_chamber,
             P_chamber):
    """Nuclear engine sizer from level 0 physics-based equation from equations
    in Space Propulsion Analysis and Design, by Ronald W. Humble, et. al.
    Estimated mass is of the engine core and related components (no tanks,
    Support structure, pressurant, etc.) This model assumes an expander cycle
    for the turbopump assembly, with redundant turbopump assemblies.

    Args
    ----
    total_thrust : float
        The thrust of the engine (kN)

    engine_thrust : float
        The thrust per engine (kN)

    isp : float
        The specific impulse of the engine (s)

    propellant : str
        The propellant of the engine

    core_type : str
        The core type, one of ['PBR','CERMET']

    T_chamber : float
        The chamber temperature of the reactor core (K)

    P_chamber : float
        The chamber pressure of the reactor core (MPa)

    Returns
    -------
    inert_mass : float
        The inert mass of the the engine system (kg)

    power_req : float
        The required power (W)
    """

    inert_mass = 0.
    power_req = 0.
    G0.convert_to_unit('m/s**2')

    if propellant.count('/') != 0:
        msg = ('may only specify a single propellant for nuclear engines')
        raise Exception(msg)

    if not fluids.check_def(propellant):
        msg = ('"{0}" is not a defined fluid'.format(propellant))
        raise Exception(msg)
```

```
# input checking
if type(core_type) != str:
    msg = ('core_type must be a string')
    raise Exception(msg)
else:
    if core_type.lower() not in ['pbr','cermet']:
        msg = ("invalid core type, must be one of ['PBR','CERMET']")
        raise Exception(msg)

if P_chamber < 1 or P_chamber > 10:
    msg = ('chamber pressure, P_chamber, must be in the range [1:10]')
    raise Exception(msg)

num_cores = ceil(total_thrust / engine_thrust)
m_dot = engine_thrust * 1000. / G0.value / isp # kg/s
T0 = fluids.tvap(propellant,P_chamber)
hvap = fluids.get_property(propellant,'Hvap') * 1000.
# From Eq. 8.8 in SPAD
P_core = m_dot * (hvap + quad(_prop_cp,T0,T_chamber,args=(propellant,))[0]) / 1e6 # MW

if P_core > 2000.:
    msg = ('Core power must be less than 2000 MW. '
           'Try decreasing "engine_thrust" and/or "T_chamber"')
    raise Exception(msg)

# reactor dimensions and mass, Eq. 8.44 - 8.49 in SPAD
if core_type == 'pbr':
    rho_core = 1600. # kg/m^3
    if P_core < 250:
        # 7 element core
        R_core = 9.0958e-10*P_core**4 - 1.3261e-6*P_core**3 + 7.1665e-4*P_core**2 - 0.1735*P_core
        ↪  + 47.625 # cm
        H_core = -0.000283*P_core**2 + 0.5203*P_core + 26.06 # cm
    elif P_core > 750:
        # 37 element core
        R_core = 4.905e-11*P_core**4 - 2.881e-7*P_core**3 + 6.2522e-4*P_core**2 - 0.5992*P_core +
        ↪  252.28 # cm
        H_core = -4.027e-5*P_core**2 + 0.1427*P_core + 17.9883 # cm
    else:
        # 19 element core
        R_core = -2.655e-12*P_core**5 + 8.946e-9*P_core**4 - 1.1703e-5*P_core**3
        ↪  +7.427e-3*P_core**2 - 2.2955*P_core + 313.34 # cm
        H_core = -6.502e-6*P_core**2 + 0.05009*P_core + 18.335 # cm
elif core_type == 'cermet':
    rho_core = 8500. # kg/m^3
    R_core = 0.0034*P_core + 20.79
    H_core = 0.0067*P_core + 41.418
R_core = R_core / 100. # cm -> m
H_core = H_core / 100. # cm -> m

V_core = pi * R_core**2 * H_core
m_core = num_cores * rho_core * V_core

# nozzle calculations
# thermochemistry
R_spec = fluids.rspec(propellant)
heat_ratio = fluids.heat_ratio(propellant,T_chamber)
# Eq. 5.12 in SPAD for characteristic exhaust velocity (m/s)
combustion_efficiency = 0.999
nozzle_efficiency = 0.99
P_amb = 0. # (MPa) assumes in-space engine
# c_star in (m/s)
c_star = (combustion_efficiency * sqrt(heat_ratio*R_spec*T_chamber) /
          (heat_ratio*(2/(heat_ratio + 1))**((heat_ratio + 1)/(2*heat_ratio - 2))))
isp_max = nozzle_efficiency * (c_star * heat_ratio / G0.value * sqrt((2 / (heat_ratio - 1))*
                              (2 / (heat_ratio + 1))**((heat_ratio + 1)/(heat_ratio - 1))))
isp_max = floor(0.98 * isp_max)
isp_min = ceil(isp_max * (1 - 0.3))
```

```
# check isp input for feasibility based on flow parameters. This is because
# the mach calculation is very sensitive to the target isp
if isp > isp_max or isp < isp_min:
    msg = ('isp is outside of feasible bounds: [{0}:{1}] s'.format(isp_min,isp_max))
    raise Exception(msg)

mach = brentq(_mach_calc,1.,100.,args=(isp,P_chamber,P_amb,heat_ratio,c_star,nozzle_efficiency))
expansion_ratio = (1/mach) * ((2 / (heat_ratio + 1))*(1 + ((heat_ratio - 1)/2 *
↪  mach**2)))**((heat_ratio + 1)/(heat_ratio - 1))

A_throat = m_dot * c_star / (P_chamber * 1e6) # Equation 5.10 from SPAD, (m)
d_throat = 2*sqrt(A_throat/pi)
d_exit = 2*sqrt(expansion_ratio*A_throat/pi)

throat_thick = 3. * P_chamber * (d_throat/2.) / 310. # assumes a material Ut = 310 MPa
l_nozz = (d_exit - d_throat) / (2*tan(0.261799)) # assumes nozzle half angle of 15 deg (0.261799
↪  rad)

et_ratio = 1. # exit thickness ratio, assumed 0 -> nozzle exit wall thickness = 0
x1 = ((et_ratio*throat_thick)-throat_thick)/l_nozz
x2 = (0.5*(d_exit-d_throat))/l_nozz
# assumes a material density of 8500. kg/m^3
m_nozz =
↪  num_cores*2*pi*8500.*l_nozz*(((1/3)*x1*x2*(l_nozz**2))+(((0.5*(x1*0.5*d_throat))+(0.5*(x2*throat_thick)))*l_noz

V_vessel = V_core + ((4/3) * pi * R_core**3) # core volume + hemispherical end caps
# uses the Pv/W method for calculating a pressure vessel mass, sec. 5.4.4 of SPAD
m_vessel = 2 * (P_chamber * 1e6) * V_vessel / G0.value / 2500. # assumes a tank factor of 2500 for
↪  all metallic tank

# cooling + feed system
# estimates based on 40% for nozzle and combustion chamber, 35.1% for cooling, and 24.9% for
↪  injector/feed
# these numbers are from SPAD p. 504 in the nuclear engine case study
m_thrust_chamber = (m_nozz + m_vessel) / 0.4 # kg
m_feed = num_cores * m_thrust_chamber * 0.249 # kg
m_cool = num_cores * m_thrust_chamber * 0.351 # kg

# shield calculations
shield_area = pi * R_core**2
shied_aerial_density = 3500. # kg/m2, based on baseline in sec. 8.5 of SPAD
m_shield = num_cores * shied_aerial_density * shield_area

# pump calculations
num_tpas = 2 # redundant tpa
rho = fluids.density(propellant)
# this is a fit of data for the turbine/pump power balance from Larry's
# spread sheet. It assumes a fixed turbine inlet temperature of 315.5 K
turbine_inlet_temp = 315.5 # K, assumed, provides longest turbine life and most conservative mass
↪  estimates
turbine_efficiency = 0.70 # assumed
pump_efficiency = 0.75 # assumed pump efficiency
Cp = fluids.cp(propellant,turbine_inlet_temp)
heat_ratio = fluids.heat_ratio(propellant,turbine_inlet_temp)
args = (turbine_inlet_temp, turbine_efficiency,pump_efficiency,Cp,
        heat_ratio, P_chamber,m_dot,rho)
turbine_pressure_ratio = brentq(_tpa_power_ballance,1.,2.5,args=args)
# 1.2 and 1.05 are factors for pressure drop in reactor and cooling, 0.05 is pressure drop in the
↪  feed lines
delta_P = 1.2 * (1.05 * P_chamber * turbine_pressure_ratio) + 0.05
pump_head = delta_P * 1e6 / G0.value / rho
pump_power = pump_head * m_dot * G0.value / pump_efficiency # W, f(m_dot,P_chamber)
Q_dot = m_dot / rho
pump_speed = 2 * pump_head**.75 / sqrt(Q_dot) # rad/s, f(P_chamber)
tau = pump_power / pump_speed # N-m
A = 2.6; B = 0.667 # maximum values for parameters from SPAD, p. 266
m_tpa = num_tpas * 1.25 * A*tau**B # kg, adds 25% additional mass for spool up
```

```
    inert_mass = m_core + m_nozz + m_vessel + m_feed + m_cool + m_shield + m_tpa # kg

    return inert_mass, power_req

def _tpa_power_balance(turbine_pressure_ratio, turbine_inlet_temp,
                       turbine_efficiency, pump_efficiency, Cp, heat_ratio,
                       P_chamber, m_dot, rho):
    """This power balance for the turbopump assembly (TPA) assumes a pump and
    turbine in series with the reactor core (expander cycle), meaning all of
    the propellant flows through both the pump and turbine.
    """

    turbine_power = m_dot * turbine_efficiency * turbine_inlet_temp * Cp * (1 -
    ↪  (1/turbine_pressure_ratio)**((heat_ratio - 1)/heat_ratio))
    delta_P = 1.2 * (1.05 * P_chamber * turbine_pressure_ratio) + 0.05
    pump_head = delta_P * 1e6 / G0.value / rho
    pump_power = pump_head * m_dot * G0.value / pump_efficiency # W, f(m_dot,P_chamber)

    return turbine_power - pump_power

def _mach_calc(mach, isp_target, P_chamber, P_amb, heat_ratio, c_star,
               nozzle_efficiency):
    """This function is for estimating the mach of the propellant flow
    through the nozzle by matching the desired isp of the engine
    """
    if mach < 0:
        mach = 0.1

    P_chamber = P_chamber * 1e6 # Pa
    P_amb = P_amb * 1e6 # Pa

    P_exit = P_chamber * (1 + ((heat_ratio - 1)/2 * mach**2))**(heat_ratio / (1 - heat_ratio))
    expansion_ratio = (1/mach) * ((2 / (heat_ratio + 1))*(1 + ((heat_ratio - 1)/2 *
    ↪  mach**2)))**((heat_ratio + 1)/(2*heat_ratio - 2))
    isp_calc = nozzle_efficiency * (c_star * heat_ratio / G0.value * sqrt((2 / (heat_ratio - 1))*
                                    (2 / (heat_ratio + 1))**((heat_ratio + 1)/(heat_ratio - 1)) *
                                    (1 - (P_exit/P_chamber)**((heat_ratio - 1) / heat_ratio))) +
                                    c_star * expansion_ratio * (P_exit - P_amb) / G0.value /
                                    ↪  P_chamber)

    return isp_calc - isp_target

def _prop_cp(T,prop):

    return fluids.cp(prop,T)
```

## I.3 Power SubElement Model

```
# -*- coding: utf-8 -*-
"""
Description:
    A DYREQT Power subelement for Douglas Trent's PhD

Written by:
    Douglas J. Trent
    NASA Marshall Space Flight Center
    Advanced Concept Office
    douglas.trent@nasa.gov

Created: 04/06/2017
Revised: 07/31/2017
"""
# import DYREQT Subelement base class
from SubElements import SubElement
from numpy import cos
```

```
# create the structures subelement
class PowerPhD(SubElement):
    """Estimates the mass of a power subsystem, including structures where
    needed (solar arrays)

    Input Params
    ------------
    generator_type : str('pv')
        The type of power generation for the element, one of ['pv','rtg']

    transmission_efficiency : float(0.9)
        The efficiency of transmitting power from the generator/power storage
        to the load.

    cell_efficiency : float(0.148)
        The efficiency of the PV cell at converting solar energy to electrical
        energy. A value between 0.0 and 1.0

    cell_degradation : float(3.75 %/year)
        The decrease in cell power production (%/year)

    array_density : float(3.5 kg/m**2)
        The aerial density of the solar array, including PV cells and array
        structures (kg/m**2)

    discharge_depth : float(0.2)
        The depth of discharge of the battery storage system as a fraction of
        the storage capacity

    storage_specific_energy : float(30 W*h/kg)
        The energy density of the power storage system (W*h/kg)

    energy_tracking : str('peak-tracking')
        The energy tracking scheme for the solar array system. One of
        ['direct,'peak-tracking'].

    low_array_degradation : bool(False)
        If True, reduces solar array degradation due to assembly and
        configuration (shadowing), and operational temperature, reducing the
        array mass for a given power output.

    orbit_period : float(1.6467 h)
        The orbital period (h)

    max_eclipse : float(0.588 h)
        The maximum eclipse time during the mission for which batteries must
        provide the power_required (h)

    ops_distance : float(1 AU)
        The solar distance from the sun of the worst operational environment (AU)

    mission_duration : float(5 years)
        The duration of the mission (y)

    Inherited Params
    ----------------
    power_req : array
        The power required from each element subsystems (W)

    Outputs
    -------
    inert_mass : float
        The inert mass of the subelement (kg)

    heat_loads : array
        The amount of heat generated by other subsystems to be dissipated by
        the radiators (W)
```

```
"""

def __init__(self,**kwargs):
    super().__init__(**kwargs)
    # user model inputs
    self.add_param(self.base_name+'_generator_type', val=str('pv'))
    self.add_param(self.base_name+'_transmission_efficiency', val=float(0.9))
    self.add_param(self.base_name+'_cell_efficiency', val=float(0.148))
    self.add_param(self.base_name+'_cell_degredation', val=float(3.75), units='1/yr')
    self.add_param(self.base_name+'_array_density', val=float(3.5), units='kg/m**2')
    self.add_param(self.base_name+'_discharge_depth', val=float(0.2))
    self.add_param(self.base_name+'_storage_specific_energy', val=float(30), units='W*h/kg')
    self.add_param(self.base_name+'_energy_transfer', val=str('peak-tracking'))
    self.add_param(self.base_name+'_low_array_degridation', val=False)
    self.add_param(self.base_name+'_orbit_period', val=float(1.6467), units='h') # will eventually
    ↪  be inherited from DYREQT mission
    self.add_param(self.base_name+'_max_eclipse', val=float(0.588), units='h') # will eventually
    ↪  be inherited from DYREQT mission
    self.add_param(self.base_name+'_ops_distance', val=float(1), units='AU') # will eventually be
    ↪  inherited from DYREQT mission
    self.add_param(self.base_name+'_mission_duration', val=float(5), units='yr') # will eventually
    ↪  be inherited from DYREQT mission
    # outputs used by the parent element
    self.add_output(self.base_name+'_heat_load', val=float(0), units='W')
    self.add_output(self.base_name+'_inert_mass', val=float(0), units='kg')
    # power requirement from all other subelements in the parent element
    for subnum in range(self.element.num_subelements):
        if subnum != self.subelement_num:
            self.add_param('element{0}sub{1}_power_req'.format(self.element.element_num,subnum),
            ↪  val=float(0), units='W')

def pre_setup(self,problem):

    self.add_param(self.base_name+'_engine_power_req', val=0., units='W')
    for subelement in self.element.components():
        if isinstance(subelement,SubElement):
            if 'engine' in subelement.subelement_type.lower():

                ↪  self.element.connect(self.base_name+'_engine_power_req','element'+str(self.element.element_num)

def post_setup(self, problem):
    """Check inputs after final connections have been made
    """

    generator_type = self.params[self.base_name+'_generator_type'].lower()
    transmission_efficiency = self.params[self.base_name+'_transmission_efficiency']
    cell_efficiency = self.params[self.base_name+'_cell_efficiency']
    cell_degradation = self.params[self.base_name+'_cell_degredation'] / 100.
    discharge_depth = self.params[self.base_name+'_discharge_depth']
    ops_distance = self.params[self.base_name+'_ops_distance']
    orbit_period = self.params[self.base_name+'_orbit_period']
    max_eclipse = self.params[self.base_name+'_max_eclipse']
    mission_duration = self.params[self.base_name+'_mission_duration']

    if generator_type not in ['pv','rtg','none']:
        msg = ("undefined generator type. must be one of ['pv','rtg','none']")
        raise Exception(msg)

    if orbit_period < max_eclipse:
        msg = ('orbit period must be greater than the max eclipse duration')
        raise Exception(msg)

    for name,value in {'cell_efficiency':cell_efficiency,
                       'cell_degradation':cell_degradation,
                       'discharge_depth':discharge_depth,
                       'transmission_efficiency':transmission_efficiency}.items():
        if value < 0 or value > 1:
            msg = ('{0} must be a value between zero and 1'.format(name))
```

```
                raise Exception(msg)

        for name,value in {'ops_distance':ops_distance,
                           'orbit_period':orbit_period,
                           'max_eclipse':max_eclipse,
                           'mission_duration':mission_duration}.items():
            if value < 0:
                msg = ('{0} must be a positive value'.format(name))
                raise Exception(msg)

    def solve_nonlinear(self, params, unknowns, resids):

        # unpack params to local variables
        generator_type = params[self.base_name+'_generator_type'].lower()
        transmission_efficiency = params[self.base_name+'_transmission_efficiency']
        cell_efficiency = params[self.base_name+'_cell_efficiency']
        cell_degradation = params[self.base_name+'_cell_degradation']
        array_density = params[self.base_name+'_array_density']
        low_array_degradation = params[self.base_name+'_low_array_degradation']
        discharge_depth = params[self.base_name+'_discharge_depth']
        storage_specific_energy = params[self.base_name+'_storage_specific_energy']
        energy_transfer = params[self.base_name+'_energy_transfer']
        ops_distance = params[self.base_name+'_ops_distance']
        orbit_period = params[self.base_name+'_orbit_period']
        max_eclipse = params[self.base_name+'_max_eclipse']
        mission_duration = params[self.base_name+'_mission_duration']
        P_engine = params[self.base_name+'_engine_power_req']

        power_required = 0.
        for subnum in range(self.element.num_subelements):
            if subnum != self.subelement_num:
                power_required +=
                ↪  params['element{0}sub{1}_power_req'.format(self.element.element_num,subnum)]

        Pe = Pd = power_required # (W)
        Te = max_eclipse # (h)
        n = transmission_efficiency # transmission efficiency from battery/generator to load

        # size the generator
        gen_mass = 0.
        if generator_type == 'pv': # PV generator
            To = orbit_period # (h)
            Td = To - Te # (h)
            if energy_transfer == 'direct':
                Xbase = 0.85 # from Space Mission Engineering: The New SMAD, sec. 21.2.2
            else:
                Xbase = 0.8 # from Space Mission Engineering: The New SMAD, sec. 21.2.2
            Xe = Xbase*n # accounts for losses for power coming from batteries
            Xd = Xbase

            Psa = (Pe*Te/Xe + Pd*Td/Xd)/Td # (W), Space Mission Engineering: The New SMAD, eq. 21-6

            solar_flux = 1368.0 * (1/ops_distance**2) # W/m^2 # inverse square law relation
            # from Space Mission Engineering: The New SMAD, Fig. 21-24

            Po = solar_flux * cell_efficiency

            # can bring in degradation factors as technology inputs
            if low_array_degradation:
                Id = 0.95 * 0.95 # inherent degradation due to design, assembly, and thermal
            else:
                Id = 0.85 * 0.85 # inherent degradation due to design, assembly, and thermal
            # cycling. Space Mission Engineering: The New SMAD, Sec. 21.2.2, Step 4, Table 21-14

            theta = 0. # radians, assume orbit plane equals solar ecliptic plane
            # (i.e. no incidence angle losses)

            P_BOL = Po * Id * cos(theta) # Power beginning of life (W/m^2)
            D = cell_degradation / 100.

            L = mission_duration

            Ld = (1 - D)**L # life degradation

            P_EOL = P_BOL * Ld # end of life power (W/m^2)

            Asa = Psa / P_EOL # array surface area (m^2)

            Msa = Asa * array_density # solar array mass (kg)

            gen_mass = Msa # kg

        elif generator_type == 'rtg': # RTG generator
            gen_mass = 0.3801 * power_required * (2 - n) # kg
            # this is a linear curve fit of data from Element of Spacecraft Design
            # by Charles D. Brown, Sec 6.3 Table 6.13, p. 350

        # size the power storage
        storage_mass = 0. # kg

        if generator_type != 'rtg': # using solar arrays, need storage during eclipse periods
            N = 1 # number of batteries required
            DoD = discharge_depth

            batt_capacity = (Pe-P_engine)*Te / (DoD*N*n) # (W-hr)

            batt_mass = batt_capacity / storage_specific_energy # kg

            storage_mass = batt_mass * (N+1) # kg, redundancy for battery failure

        # size regulation/distribution
        # this is just an estimate based on the general mass breakdown of power
        # systems provided in Space Mission Engineering: The New SMAD, Fig. 21-6, p.641
        reg_dist_mass = (0.17/0.83) * (gen_mass + storage_mass) # kg

        inert_mass = gen_mass + storage_mass + reg_dist_mass # kg

        heat_load = power_required * (1 - n) # W, heat from power regulation/distribution

        # assign unknowns
        unknowns[self.base_name+'_heat_load'] = heat_load
        unknowns[self.base_name+'_inert_mass'] = inert_mass
```

## I.4 Structures SubElement Model

```
# -*- coding: utf-8 -*-
"""
Description:
    A DYREQT Structures subelement for Douglas Trent's PhD

Written by:
    Douglas J. Trent
    NASA Marshall Space Flight Center
    Advanced Concept Office
    douglas.trent@nasa.gov

Created: 04/06/2017
Revised: 07/31/2017
"""
# import DYREQT Subelement base class
from SubElements import SubElement
from numpy import pi, log
```

```
# create the structures subelement
class StructuresPhD(SubElement):
    """Estimates the structural mass of an element. Sizing is based on the
    design-envelope area of the element, which is estimated based on the size
    of the propellant tanks. The relation for structure mass to design-envelope
    area is from NASA document JSC-26098, "Mass Estimating and Forecasting for
    Aerospace Vehicles Based on Historical Data" by Willie Heineman, Jr.

    Input Params
    ------------
    A_de : float(1 m**2)
        The design-envelop area of the structure (m**2). This is the surface
        area of the design-envelop volume of the element. If this value is
        provided all other inputs will be ignored. This acts as a static
        override for the scaling of the structures.

    manned : bool(False)
        If the element is a manned vehicle. Increases the leading coefficient
        in the relationship to increase overall mass.

    composite : bool(False)
        If True, reduces overall mass by 30% for composite structures.

    truss : bool(False)
        If True, assume a truss structure for a single tank, otherwise, assume
        an in-line tank structure. If num_tanks is greater than 1, this input is
        ignored. This input overrides the 'manned' input setting

    adapter : bool(False)
        If True, sizes an adapter instead of a primary/secondary structure.
        This setting overrides the 'manned' or 'truss' input settings

    Inherited Params
    ----------------
    num_fuel_tanks_mps : float
        The number of fuel tanks in the main propulsion system

    diameter_fuel_tanks_mps : float
        The diameter of the fuel tanks in the main propulsion system (m)

    length_fuel_tanks_mps : float
        The length of the fuel tanks in the main propulsion system (m)

    num_ox_tanks_mps : float
        The number of oxidizer tanks in the main propulsion system

    diameter_ox_tanks_mps : float
        The diameter of the oxidizer tanks in the main propulsion system (m)

    length_ox_tanks_mps : float
        The length of the oxidizer tanks in the main propulsion system (m)

    num_fuel_tanks_mps : float
        The number of fuel tanks in the main propulsion system

    diameter_fuel_tanks_rcs : float
        The diameter of the fuel tanks in the reaction control system (m)

    length_fuel_tanks_rcs : float
        The length of the fuel tanks in the reaction control system (m)

    num_ox_tanks_rcs : float
        The number of oxidizer tanks in the reaction control system

    diameter_ox_tanks_rcs : float
        The diameter of the oxidizer tanks in the reaction control system (m)

    length_ox_tanks_rcs : float
```

```
        The length of the oxidizer tanks in the reaction control system (m)

    Outputs
    -------
    inert_mass : float
        The estimated inert mass of the structures (kg). This includes
        primary and secondary structures.
    """

    def __init__(self,**kwargs):
        super().__init__(**kwargs)
        # user model inputs
        self.add_param(self.base_name+'_A_de', val=float(-1), units='m**2')
        self.add_param(self.base_name+'_manned', val=False)
        self.add_param(self.base_name+'_composite', val=False)
        self.add_param(self.base_name+'_truss', val=False)
        self.add_param(self.base_name+'_addapter', val=False)
        # parameters from the element (DYREQT internal or from other subelements)
        self.add_param(self.element.base_name+'_num_fuel_tanks_mps', val=int(0))
        self.add_param(self.element.base_name+'_diameter_fuel_tanks_mps', val=float(0), units='m')
        self.add_param(self.element.base_name+'_length_fuel_tanks_mps', val=float(0), units='m')
        self.add_param(self.element.base_name+'_num_ox_tanks_mps', val=int(0))
        self.add_param(self.element.base_name+'_diameter_ox_tanks_mps', val=float(0), units='m')
        self.add_param(self.element.base_name+'_length_ox_tanks_mps', val=float(0), units='m')
        self.add_param(self.element.base_name+'_num_fuel_tanks_rcs', val=int(0))
        self.add_param(self.element.base_name+'_diameter_fuel_tanks_rcs', val=float(0), units='m')
        self.add_param(self.element.base_name+'_length_fuel_tanks_rcs', val=float(0), units='m')
        self.add_param(self.element.base_name+'_num_ox_tanks_rcs', val=int(0))
        self.add_param(self.element.base_name+'_diameter_ox_tanks_rcs', val=float(0), units='m')
        self.add_param(self.element.base_name+'_length_ox_tanks_rcs', val=float(0), units='m')
        # outputs used by the parent element
        self.add_output(self.base_name+'_inert_mass', val=float(0), units='kg')

    def pre_setup(self,problem):
        pass

    def post_setup(self, problem):
        """Check inputs after final connections have been made
        """

        mps_tanks = (self.params[self.element.base_name+'_num_fuel_tanks_mps'] +
                     self.params[self.element.base_name+'_num_ox_tanks_mps'])
        rcs_tanks = (self.params[self.element.base_name+'_num_fuel_tanks_rcs'] +
                     self.params[self.element.base_name+'_num_ox_tanks_rcs'])
        A_de = self.params[self.base_name+'_A_de']

        if mps_tanks + rcs_tanks == 0:
            if A_de == -1:
                msg = ('must specify a design envelope area, A_de, for '
                       'element {0} with no tanks'.format(self.element.element_num))
                raise Exception(msg)

        if A_de < 0 and A_de != -1:
            msg = ('design envelope area, A_de, for element {0} must be '
                   'greater than zero'.format(self.element.element_num))
            raise Exception(msg)

    def solve_nonlinear(self, params, unknowns, resids):

        A_de = params[self.base_name+'_A_de']
        truss = params[self.base_name+'_truss']
        manned = params[self.base_name+'_manned']
        adapter = params[self.base_name+'_adapter']
        composite = params[self.base_name+'_composite']

        tank_diameters = [params[self.element.base_name+'_diameter_fuel_tanks_mps'],
                          params[self.element.base_name+'_diameter_ox_tanks_mps'],
                          params[self.element.base_name+'_diameter_fuel_tanks_rcs'],
                  params[self.element.base_name+'_diameter_ox_tanks_rcs']]

        tank_lengths = [params[self.element.base_name+'_length_fuel_tanks_mps'],
                        params[self.element.base_name+'_length_ox_tanks_mps'],
                        params[self.element.base_name+'_length_fuel_tanks_rcs'],
                        params[self.element.base_name+'_length_ox_tanks_rcs']]

        num_tanks = [params[self.element.base_name+'_num_fuel_tanks_mps'],
                     params[self.element.base_name+'_num_ox_tanks_mps'],
                     params[self.element.base_name+'_num_fuel_tanks_rcs'],
                     params[self.element.base_name+'_num_ox_tanks_rcs']]

        if A_de == -1: # no value specified by user, estimate it from tank geometries

            d_tanks = 0.
            l_tanks = 0.
            for i in range(0,2):
                if tank_diameters[i] > 0.:
                    d_tanks += tank_diameters[i]**2 / sum(tank_diameters)
                    l_tanks += tank_lengths[i]**2 /  sum(tank_lengths)

            if sum(num_tanks[0:2]) > 2: # assume disk shape
                # diameter of a circle which fits the 120% diameter tanks
                d_de = (1.2*d_tanks)*(1.1655*log(sum(num_tanks))+0.9571)
                # surface area of a cylinder with d_de and l=120% tank diameter
                A_de = 2*pi*(d_de/2)**2 + 2*pi*(d_de/2)*(1.2*l_tanks)
            elif sum(num_tanks[0:2]) == 2: # assume two stacked mps tanks
                d_de = d_tanks
                A_de = 2*pi*(d_de/2)**2 + 2*pi*(d_de/2)*(sum(num_tanks[0:2])*l_tanks)
            else: # assume truss/drop tank
                d_de = d_tanks
                if truss:
                    truss_density_factor = 0.425 # m^3/m^3
                    A_de = truss_density_factor * (pi*(d_de/2)**2 + pi*(d_de/2)*(l_tanks))
                else:
                    A_de = 2*pi*(d_de/2)**2 + 2*pi*(d_de/2)*(l_tanks)

        SF = 1.27 # scaling factor based on type
        SF = 0.71
        if manned:
            SF = 2.0
        if truss or sum(num_tanks[0:2]) == 1:
            SF = 0.71
        if adapter:
            SF = 0.99

        inert_mass = SF*(A_de*10.7639)**(1.15) # 10.7639 conversion from m^3 -> f^3

        if composite:
            inert_mass = 0.7*inert_mass

        inert_mass = inert_mass * 0.453592 # kg

        # assign unknowns
        unknowns[self.base_name+'_inert_mass'] = inert_mass
```

## I.5 Tanks SubElement Model

```
# -*- coding: utf-8 -*-
"""
Description:
    A DYREQT Tanks subelement for Douglas Trent's PhD

Written by:
    Douglas J. Trent
    NASA Marshall Space Flight Center
    Advanced Concept Office
    douglas.trent@nasa.gov

Created: 04/06/2017
Revised: 08/31/2017
"""
import sys
sys.path.append(DYREQT_DIR)
# import DYREQT Subelement base class
from SubElements import SubElement
# import other modules
import FluidsDef as fluids
from numpy import array, pi, zeros
from copy import deepcopy

# create the structures subelement
class TanksPhD(SubElement):
    """Sizes a set of tanks based on the user inputs for a propulsive stage.
    Sizing is done mostly from level-0 physics based equation, with correction
    factors for isentropic fluid expansion and composite materials.

    Input Params
    ------------
    num_fuel_tanks_mps : int(1)
        The number of main propulsion system fuel tanks.

    num_ox_tanks_mps : int(1)
        The number of main propulsion system oxidizer tanks.

    fuel_pressures_mps : float(0.3 MPa)
        The main propulsion system fuel tank pressures (MPa).

    ox_pressures_mps : float(0.3 MPa)
        The main propulsion system oxidizer tank pressures (MPa).

    ld_ratio_fuel_tanks_mps : float(1.0)
        The main propulsion system fuel tank L/D ratio.

    ld_ratio_ox_tanks_mps : float(1.0)
        The main propulsion system oxidizer tank L/D ratio.

    separator_type_mps : str('pmd','ped','')
        The type of device used to separate pressurant gas and liquid
        propellant in the main propulsion system propellant tank

    num_fuel_tanks_rcs : int(1)
        The number of reaction control system fuel tanks.

    num_ox_tanks_rcs : int(1)
        The number of reaction control system oxidizer tanks.

    fuel_pressures_rcs : float(0.3 MPa)
        The reaction control system fuel tank pressures (MPa).

    ox_pressures_rcs : float(0.3 MPa)
        The reaction control system oxidizer tank pressures (MPa).

    ld_ratio_fuel_tanks_rcs : float(1.0)
        The reaction control system fuel tank L/D ratio.

    ld_ratio_ox_tanks_rcs : float(1.0)
        The reaction control system oxidizer tank L/D ratio.

    separator_type_rcs : str('pmd','ped','')
        The type of device used to separate pressurant gas and liquid
        propellant in the reaction control system propellant tank

    pressurant : str
        The pressurant for tanks

    pressurant_pressure : float
        The initial pressure of the pressurant tanks (MPa)

    num_tanks_pressurant : int
        The number of pressurant tanks to size

    tank_ld_ratio_pressurant : float
        The L/D ratio of the pressurant tank

    material_density : float
        The density of the tank material (kg/m^3)

    material_strength : float
        The ultimate strength of the tank material (MPa)

    copv_pressurant_tanks : bool(False,True)
        If True, utilizes composite overwrap pressure vessels for the pressurant
        tanks

    composite_fuel_tanks_mps : bool
        If True, utilizes composite materials for the MPS fuel tanks
        resulting in a 30% mass reduction from Aluminum-Lithium tanks.

    composite_ox_tanks_mps : bool
        If True, utilizes composite materials for the MPS oxidizer tanks
        resulting in a 30% mass reduction from Aluminum-Lithium tanks.

    composite_fuel_tanks_rcs : bool
        If True, utilizes composite materials for the RCS fuel tanks
        resulting in a 30% mass reduction from Aluminum-Lithium tanks.

    composite_ox_tanks_rcs : bool
        If True, utilizes composite materials for the RCS oxidizer tanks
        resulting in a 30% mass reduction from Aluminum-Lithium tanks.

    ivfm : bool
        If True, assumes integrated vehicle fluid management, which will combine
        the RCS and MPS propellant systems into a single system, reducing
        the number of tanks and overall mass of the propellant storage system

    Inherited Params
    ----------------
    propellants_mps : str
        The propellants for the main propulsion system

    propellants_rcs : str
        The propellants for the reaction system

    mixture_ratio_mps : float
        The mixture ratio of the main propulsion system propellants

    mixture_ratio_rcs : float
        The mixture ratio of the reaction control system propellants

    max_propellant_mass_mps : float
        The total amount of propellant required for all impulsive maneuvers
        performed by the main propulsion system (kg)

    max_propellant_mass_rcs : float
        The total amount of propellant required for all impulsive maneuvers
        performed by the reaction control system (kg)

    propellant_mass_mps : array
        The propellant required for each impulsive maneuver the parent element
        performs (kg)
    propellant_mass_rcs : array
        The propellant required for each impulsive maneuver the parent element
        performs (kg)

    Outputs
    -------
    num_fuel_tanks_mps : float
        The number of fuel tanks in the main propulsion system

    diameter_fuel_tanks_mps : float
        The diameter of the fuel tanks in the main propulsion system (m)

    length_fuel_tanks_mps : float
        The length of the fuel tanks in the main propulsion system (m)

    pressure_fuel_tanks_mps : float
        The pressure of the fuel tanks in the main propulsion system (MPa)

    num_ox_tanks_mps : float
        The number of oxidizer tanks in the main propulsion system

    diameter_ox_tanks_mps : float
        The diameter of the oxidizer tanks in the main propulsion system (m)

    length_ox_tanks_mps : float
        The length of the oxidizer tanks in the main propulsion system (m)

    pressure_ox_tanks_mps : float
        The pressure of the oxidizer tanks in the main propulsion system (MPa)

    num_fuel_tanks_mps : float
        The number of fuel tanks in the main propulsion system

    diameter_fuel_tanks_rcs : float
        The diameter of the fuel tanks in the reaction control system (m)

    length_fuel_tanks_rcs : float
        The length of the fuel tanks in the reaction control system (m)

    pressure_fuel_tanks_rcs : float
        The pressure of the fuel tanks in the reaction control system (MPa)

    num_ox_tanks_rcs : float
        The number of oxidizer tanks in the reaction control system

    diameter_ox_tanks_rcs : float
        The diameter of the oxidizer tanks in the reaction control system (m)

    length_ox_tanks_rcs : float
        The length of the oxidizer tanks in the reaction control system (m)

    pressure_ox_tanks_rcs : float
        The pressure of the oxidizer tanks in the reaction control system (MPa)

    inert_mass : float
        The estimated inert mass of the propellant tanks (kg). This includes
        the mass of the dry tanks, miscellaneous hardware, pressurant, and
        propellant trap.
    """

    def __init__(self,**kwargs):
        super().__init__(**kwargs)
        # user model inputs
        self.add_param(self.base_name+'_num_fuel_tanks_mps', val=int(1))
        self.add_param(self.base_name+'_num_ox_tanks_mps', val=int(1))
        self.add_param(self.base_name+'_fuel_pressure_mps', val=float(0.3), units='MPa')
        self.add_param(self.base_name+'_ox_pressure_mps', val=float(0.3), units='MPa')
        self.add_param(self.base_name+'_ld_ratio_fuel_tanks_mps', val=float(1))
        self.add_param(self.base_name+'_ld_ratio_ox_tanks_mps', val=float(1))
        self.add_param(self.base_name+'_seperator_type_mps', val=str('pmd'))
        self.add_param(self.base_name+'_num_fuel_tanks_rcs', val=int(0))
        self.add_param(self.base_name+'_num_ox_tanks_rcs', val=int(0))
        self.add_param(self.base_name+'_fuel_pressure_rcs', val=float(0.3), units='MPa')
        self.add_param(self.base_name+'_ox_pressure_rcs', val=float(0.3), units='MPa')
        self.add_param(self.base_name+'_ld_ratio_fuel_tanks_rcs', val=float(1))
        self.add_param(self.base_name+'_ld_ratio_ox_tanks_rcs', val=float(1))
        self.add_param(self.base_name+'_seperator_type_rcs', val=str('pmd'))
        self.add_param(self.base_name+'_pressurant', val=str('He'))
        self.add_param(self.base_name+'_pressurant_pressure', val=float(1), units='MPa')
        self.add_param(self.base_name+'_num_tanks_pressurant', val=int(1))
        self.add_param(self.base_name+'_tank_ld_ratio_pressurant', val=float(1))
        self.add_param(self.base_name+'_material_strength', val=float(1), units='MPa')
        self.add_param(self.base_name+'_material_density', val=float(1), units='kg/m**3')
        self.add_param(self.base_name+'_copv_pressurant_tank', val=False)
        self.add_param(self.base_name+'_composite_fuel_tanks_mps', val=False)
        self.add_param(self.base_name+'_composite_ox_tanks_mps', val=False)
        self.add_param(self.base_name+'_composite_fuel_tanks_rcs', val=False)
        self.add_param(self.base_name+'_composite_ox_tanks_rcs', val=False)
        self.add_param(self.base_name+'_ivfm', val=False)
        # parameters from the element (DYREQT internal or from other subelements)
        self.add_param(self.element.base_name+'_propellants_mps', val=str())
        self.add_param(self.element.base_name+'_propellants_rcs', val=str())
        self.add_param(self.element.base_name+'_mixture_ratio_mps', val=float(1))
        self.add_param(self.element.base_name+'_mixture_ratio_rcs', val=float(1))
        self.add_param(self.element.base_name+'_max_propellant_mass_mps', val=float(0), units='kg')
        self.add_param(self.element.base_name+'_max_propellant_mass_rcs', val=float(0), units='kg')
        self.add_param(self.element.base_name+'_max_single_burn_prop_mass_rcs', val=float(0),
        ↪  units='kg')
        self.add_param(self.element.base_name+'_propellant_mass_mps',
        ↪  val=zeros(self.element.num_events+1), units='kg')
        self.add_param(self.element.base_name+'_propellant_mass_rcs',
        ↪  val=zeros(self.element.num_events+1), units='kg')
        # outputs inherited by the parent element (for use by other subelements or DYREQT)
        self.add_output(self.element.base_name+'_num_fuel_tanks_mps', val=int(1))
        self.add_output(self.element.base_name+'_diameter_fuel_tanks_mps', val=float(0), units='m')
        self.add_output(self.element.base_name+'_length_fuel_tanks_mps', val=float(0), units='m')
        self.add_output(self.element.base_name+'_pressure_fuel_tanks_mps', val=float(0.3),
        ↪  units='MPa')
        self.add_output(self.element.base_name+'_num_ox_tanks_mps', val=int(1))
        self.add_output(self.element.base_name+'_diameter_ox_tanks_mps', val=float(0), units='m')
        self.add_output(self.element.base_name+'_length_ox_tanks_mps', val=float(0), units='m')
        self.add_output(self.element.base_name+'_pressure_ox_tanks_mps', val=float(0.3), units='MPa')
        self.add_output(self.element.base_name+'_num_fuel_tanks_rcs', val=int(0))
        self.add_output(self.element.base_name+'_diameter_fuel_tanks_rcs', val=float(0), units='m')
        self.add_output(self.element.base_name+'_length_fuel_tanks_rcs', val=float(0), units='m')
        self.add_output(self.element.base_name+'_pressure_fuel_tanks_rcs', val=float(0.3),
        ↪  units='MPa')
        self.add_output(self.element.base_name+'_num_ox_tanks_rcs', val=int(0))
        self.add_output(self.element.base_name+'_diameter_ox_tanks_rcs', val=float(0), units='m')
        self.add_output(self.element.base_name+'_length_ox_tanks_rcs', val=float(0), units='m')
        self.add_output(self.element.base_name+'_pressure_ox_tanks_rcs', val=float(0.3), units='MPa')
        # outputs used by the parent element
        self.add_output(self.base_name+'_inert_mass', val=float(0), units='kg')

    def pre_setup(self,problem):
        pass

    def post_setup(self, problem):
        """Check inputs after final connections have been made
        """
        for propsys in ['mps','rcs']:

            props = self.params[self.element.base_name+'_propellants_'+propsys]
            num = props.count('/') + 1

            if num == 1 and not props:
```

```
            msg = ('must specify at least one propellant for the {0}'.format(propsys.upper()))
            raise Exception(msg)

    def solve_nonlinear(self, params, unknowns, resids):

        ivfm = deepcopy(params[self.base_name+'_ivfm'])

        if ivfm:
            if params[self.element.base_name+'_propellants_mps'] !=
            ↪  params[self.element.base_name+'_propellants_rcs']:
                ivfm = False

        # set local variables from params
        pres_tank_pressure = params[self.base_name+'_pressurant_pressure']
        num_pressurant_tanks = params[self.base_name+'_num_tanks_pressurant']
        pres_tank_ld_ratio = params[self.base_name+'_tank_ld_ratio_pressurant']
        copv_pressurant_tanks = params[self.base_name+'_copv_pressurant_tank']
        material_strength = params[self.base_name+'_material_strength']
        material_density = params[self.base_name+'_material_density']

        inert_mass = []
        for propsys in ['mps','rcs']:

            # set local variables from params
            pressurant = params[self.base_name+'_pressurant']
            if propsys == 'mps':
                if ivfm:
                    usable_prop = (params[self.element.base_name+'_max_propellant_mass_mps'] +
                                   params[self.element.base_name+'_max_propellant_mass_rcs'])
                    burn_props = (params[self.element.base_name+'_propellant_mass_mps'] +
                                  params[self.element.base_name+'_propellant_mass_rcs'])
                else:
                    usable_prop = params[self.element.base_name+'_max_propellant_mass_mps']
                    burn_props = params[self.element.base_name+'_propellant_mass_mps']
                propellants = params[self.element.base_name+'_propellants_mps']
                mixture_ratio = params[self.element.base_name+'_mixture_ratio_mps']
                separator_type = params[self.base_name+'_separator_type_mps']
                num_ox_tanks = params[self.base_name+'_num_ox_tanks_mps']
                num_fuel_tanks = params[self.base_name+'_num_fuel_tanks_mps']
                ox_tank_pressure = params[self.base_name+'_ox_pressure_mps']
                fuel_tank_pressure = params[self.base_name+'_fuel_pressure_mps']
                ox_tank_ld_ratio = params[self.base_name+'_ld_ratio_ox_tanks_mps']
                fuel_tank_ld_ratio = params[self.base_name+'_ld_ratio_fuel_tanks_mps']
                composite_ox_tanks = params[self.base_name+'_composite_ox_tanks_mps']
                composite_fuel_tanks = params[self.base_name+'_composite_fuel_tanks_mps']
                # assign static outputs
                unknowns[self.element.base_name+'_num_fuel_tanks_mps'] = num_fuel_tanks
                unknowns[self.element.base_name+'_pressure_fuel_tanks_mps'] = fuel_tank_pressure
                unknowns[self.element.base_name+'_num_ox_tanks_mps'] = num_ox_tanks
                unknowns[self.element.base_name+'_pressure_ox_tanks_mps'] = ox_tank_pressure
            elif propsys == 'rcs':
                if ivfm:
                    usable_prop = 0.
                    num_ox_tanks = 0.
                    num_fuel_tanks = 0.
                else:
                    usable_prop = params[self.element.base_name+'_max_propellant_mass_rcs']
                    num_ox_tanks = params[self.base_name+'_num_ox_tanks_rcs']
                    num_fuel_tanks = params[self.base_name+'_num_fuel_tanks_rcs']
                burn_props = params[self.element.base_name+'_propellant_mass_rcs']
                propellants = params[self.element.base_name+'_propellants_rcs']
                mixture_ratio = params[self.element.base_name+'_mixture_ratio_rcs']
                separator_type = params[self.base_name+'_separator_type_rcs']
                ox_tank_pressure = params[self.base_name+'_ox_pressure_rcs']
                fuel_tank_pressure = params[self.base_name+'_fuel_pressure_rcs']
                ox_tank_ld_ratio = params[self.base_name+'_ld_ratio_ox_tanks_rcs']
                fuel_tank_ld_ratio = params[self.base_name+'_ld_ratio_fuel_tanks_rcs']
                composite_ox_tanks = params[self.base_name+'_composite_ox_tanks_rcs']
                composite_fuel_tanks = params[self.base_name+'_composite_fuel_tanks_rcs']
                # assign static outputs
                unknowns[self.element.base_name+'_num_fuel_tanks_rcs'] = num_fuel_tanks
                unknowns[self.element.base_name+'_pressure_fuel_tanks_rcs'] = fuel_tank_pressure
                unknowns[self.element.base_name+'_num_ox_tanks_rcs'] = num_ox_tanks
                unknowns[self.element.base_name+'_pressure_ox_tanks_rcs'] = ox_tank_pressure

            prop_trap = 0.01 * usable_prop
            sized_prop = usable_prop + prop_trap
            props_list = propellants.replace(' ','').lower().split('/')

            ox = None
            fuel = None
            if len(props_list) >1:
                ox = props_list[0]
                fuel = props_list[1]
            else:
                fuel = props_list[0]

            # perform some input checks
            prop_types = []
            for i,propellant in enumerate([fuel,ox]):
                if propellant:
                    prop_types.append(None)
                    if propellant == 'solid':
                        prop_types[i] = 'solid'
                        prop_trap = 0.
                    else:
                        res,fluid_def = fluids.check_def(propellant)
                        if res:
                            prop_types[i] = fluid_def['type']
                        else:
                            msg = ('"{0}" is not a defined fluid'.format(propellant))
                            raise Exception(msg)

            if len(prop_types) > 1 and any(True for x in prop_types if x == 'electric'):
                msg = ('may only select a single propellant when selecting an electric '
                       'engine propellant')
                raise Exception(msg)

            if pressurant:
                if pressurant.lower() != 'none':
                    if not fluids.check_def(pressurant):
                        raise Exception('"{0}" is not a defined fluid'.format(pressurant))
                else:
                    pressurant = ''
            else:
                pressurant = ''

            if pressurant:
                if pres_tank_pressure <= 2*max([fuel_tank_pressure,ox_tank_pressure]):
                    msg = ('pressurant tank initial pressure must be at least twice '
                           'the maximum propellant tank operating pressure')
                    raise Exception(msg)

            for prop_type in prop_types:
                if prop_type != 'cryogenic':
                    pressurant = 'He'

            if separator_type:
                if separator_type.lower() != 'none':
                    if separator_type.lower() not in ['pmd','ped']:
                        msg = ('separator_type must be one of "{0}"'.format())
                        raise Exception(msg)
                else:
                    separator = ''
            else:
                separator = ''
```

```
ox_volume = 0.
ox_tank_mass = 0.
fuel_volume = 0.
fuel_tank_mass = 0.
pressurant_tank_mass = 0.
pressurant_mass = 0.
AL2195_density = 2685.0 # kg/m^3
AL2195_strength = 628.6 # MPa

# determine the mass of fuel and oxidizer
if fuel and ox:
    fuel_mass = sized_prop / (mixture_ratio + 1)
    ox_mass = (sized_prop*mixture_ratio) / (mixture_ratio + 1)
else:
    fuel_mass = sized_prop
    ox_mass = 0.

# size tanks
if fuel and num_fuel_tanks and prop_types[0] != 'solid':
    # determine fuel volume
    if prop_types[0] == 'electric':
        fuel_density = fluids.density(fuel,liquid=False,P=fuel_tank_pressure) # kg/m^3
    else:
        fuel_density = fluids.density(fuel) # kg/m^3
    fuel_volume = fuel_mass / fuel_density # m^3
    if not pressurant and prop_types[0] != 'electric':
        # add volume for autogenous pressurization
        fuel_density_gas = fluids.density(fuel, liquid=False, P=fuel_tank_pressure)
        res = fuel_volume
        fuel_volume_new = 0.
        pressurization_fuel = 1.
        while pressurization_fuel > 1e-3:
            pressurization_fuel = res * fuel_density_gas
            fuel_mass += pressurization_fuel
            fuel_volume_new = fuel_mass/fuel_density
            res = abs(fuel_volume_new - fuel_volume)
            fuel_volume = fuel_volume_new
    fuel_volume = fuel_volume * 1.05 # 5% ullage
    # design the tank
    volume = fuel_volume/num_fuel_tanks
    ld_ratio = fuel_tank_ld_ratio
    pressure = fuel_tank_pressure
    separator = separator_type
    if composite_fuel_tanks:
        density = AL2195_density
        strength = AL2195_strength
        fuel_tank_mass,fuel_tank_radius =
        ↪  self._tank_mass(density,strength,volume,ld_ratio,
                                                    pressure,separator,1.5)
        fuel_tank_mass = 0.7*fuel_tank_mass
    else:
        density = material_density
        strength = material_strength
        fuel_tank_mass,fuel_tank_radius =
        ↪  self._tank_mass(density,strength,volume,ld_ratio,
                                                    pressure,separator,1.5)
    # assign calculated outputs
    if propsys == 'mps':
        unknowns[self.element.base_name+'_diameter_fuel_tanks_mps'] = 2*fuel_tank_radius
        unknowns[self.element.base_name+'_length_fuel_tanks_mps'] =
        ↪  2*fuel_tank_radius*fuel_tank_ld_ratio
    elif propsys == 'rcs':
        unknowns[self.element.base_name+'_diameter_fuel_tanks_rcs'] = 2*fuel_tank_radius
        unknowns[self.element.base_name+'_length_fuel_tanks_rcs'] =
        ↪  2*fuel_tank_radius*fuel_tank_ld_ratio

if ox and num_ox_tanks:
    # determine ox volume
    if prop_types[1] == 'electric':
        ox_density = fluids.density(ox,liquid=False,P=ox_tank_pressure) # kg/m^3
    else:
        ox_density = fluids.density(ox) # kg/m^3
    ox_volume = ox_mass / ox_density # m^3
    if not pressurant and prop_types[0] != 'electric':
        # add volume for autogenous pressurization
        ox_density_gas = fluids.density(ox, liquid=False, P=ox_tank_pressure)
        res = ox_volume
        ox_volume_new = 0.
        pressurization_ox = 1.
        while pressurization_ox > 1e-3:
            pressurization_ox = res * ox_density_gas
            ox_mass += pressurization_ox
            ox_volume_new = ox_mass/ox_density
            res = abs(ox_volume_new - ox_volume)
            ox_volume = ox_volume_new
    ox_volume = ox_volume * 1.05 # 5% ullage
    # design the tank
    volume = ox_volume/num_ox_tanks
    ld_ratio = ox_tank_ld_ratio
    pressure = ox_tank_pressure
    separator = separator_type
    if composite_ox_tanks:
        density = AL2195_density
        strength = AL2195_strength
        ox_tank_mass,ox_tank_radius = self._tank_mass(density,strength,volume,ld_ratio,
                                                      pressure,separator,1.5)
        ox_tank_mass = 0.7*fuel_tank_mass
    else:
        density = material_density
        strength = material_strength
        ox_tank_mass,ox_tank_radius = self._tank_mass(density,strength,volume,ld_ratio,
                                                      pressure,separator,1.5)
    # assign calculated outputs
    if propsys == 'mps':
        unknowns[self.element.base_name+'_diameter_ox_tanks_mps'] = 2*ox_tank_radius
        unknowns[self.element.base_name+'_length_ox_tanks_mps'] =
        ↪  2*ox_tank_radius*ox_tank_ld_ratio
    elif propsys == 'rcs':
        unknowns[self.element.base_name+'_diameter_ox_tanks_rcs'] = 2*ox_tank_radius
        unknowns[self.element.base_name+'_length_ox_tanks_rcs'] =
        ↪  2*ox_tank_radius*ox_tank_ld_ratio

if pressurant and not (ivfm and propsys == 'rcs'):
    # determine pressurant mass and volume
    prop_tank_pressure = max([fuel_tank_pressure,ox_tank_pressure])
    pres_density_init = fluids.density(pressurant, liquid=False, P=pres_tank_pressure)
    isentropic = False
    if usable_prop:
        if max(burn_props) / usable_prop > 0.1: # more than 10% total prop in a single
        ↪  burn
            isentropic = True
    if isentropic:
        CF = (2*prop_tank_pressure/pres_tank_pressure)**0.2227
        pres_density_final = fluids.density(pressurant, liquid=False,
        ↪  P=prop_tank_pressure, T=CF*293.0)
    else:
        pres_density_final = fluids.density(pressurant, liquid=False,
        ↪  P=prop_tank_pressure)
    pres_density_res = fluids.density(pressurant, liquid=False, P=prop_tank_pressure*2)
    pres_mass = (fuel_volume + ox_volume) * pres_density_final
    new_pres_mass = pres_mass
    old_pres_mass = 0.
    res = 1.
    while res > 1e-3:
        pres_volume = new_pres_mass / pres_density_init # m^3
```

```
                res_pres_mass = pres_volume * pres_density_res
                new_pres_mass = pres_mass + res_pres_mass
                res = abs(old_pres_mass - new_pres_mass)
                old_pres_mass = new_pres_mass
            pres_mass = new_pres_mass * 1.1 # add a contingency of 10% mass
            pressurant_mass = pres_mass
            pres_volume = pres_mass / pres_density_init
            # design the tank
            volume = pres_volume/num_pressurant_tanks
            density = material_density
            strength = material_strength
            ld_ratio = pres_tank_ld_ratio
            pressure = pres_tank_pressure
            separator = ''
            pressurant_tank_mass,_ = self._tank_mass(density,strength,volume,ld_ratio,
                                                    pressure,separator,1.5)

        # modify tank masses for autogenous pressurization
        if prop_types[0].lower() != 'electric' and not pressurant:
            fuel_tank_mass = 1.1*fuel_tank_mass
            ox_tank_mass = 1.1*ox_tank_mass

        if copv_pressurant_tanks:
            pressurant_tank_mass = 0.7*pressurant_tank_mass

        total_tank_mass = (ox_tank_mass * num_ox_tanks +
                           fuel_tank_mass * num_fuel_tanks +
                           pressurant_tank_mass * num_pressurant_tanks)

        miscellaneous_hardware = 0.15 * total_tank_mass # plumbing, brackets, insulation,

        if ivfm and propsys == 'rcs':
            # mass for accumulator pumps (15 kg each, 2 for each accumulator tank)
            # and 15% for associated hardware/plumbing
            miscellaneous_hardware += 1.15*(bool(num_ox_tanks)+bool(num_fuel_tanks))*2*15.

        inert_mass.append(total_tank_mass + miscellaneous_hardware + pressurant_mass + prop_trap)

    # assign calculated outputs
    unknowns[self.base_name+'_inert_mass'] = sum(inert_mass)

def _tank_mass(self,material_density,material_strength,volume,ld_ratio,
               tank_pressure,separator_type,safety_factor):
    """Calculates the mass of a tank based on a volume, pressure, and material
    properties. Including assumptions for weld lands, inlet/outlet flanges,
    structural attach points, and a separator device, when required.

    Args
    ----
    material_density : float
        Density of the tank material (kg/m^3)

    material_strength : float
        Ultimate strength of the tank material (MPa)

    volume : float
        Volume of the tank (m^3)

    ld_ratio : float
        Ratio of the length over the diameter of the tank

    tank_pressure : float
        Pressure in the tank (MPa)

    separator_type : str
        The type of separator used in the tank, one of ["pmd","ped"]

    safety_factor : float
```

```
        Safety factor on the stress in the tank

    Returns
    -------
    dry_mass : float
        The dry mass of the sized tank (kg)

    r : float
        The tank outer radius (m)
    """

    dry_mass=0.
    r = 0.
    t = 0.
    if volume != 0:
        # bare tank mass
        if ld_ratio >= 1: # sphere or capsule
            r = 0.5*(12*volume/(pi*(3*ld_ratio-1)))**(1/3)
        else: # spherical caps
            r = 0.5*(24*volume/(pi*ld_ratio*(3 + ld_ratio**2)))**(1/3)
        t = tank_pressure*safety_factor*r/(2*material_strength)
        if t < 0.000254: # m
            t = 0.000254
        if ld_ratio >= 1:
            endcaps_mass = 4/3.*pi*((r+t)**3-r**3)*material_density
        else:
            A = r+t
            H = ld_ratio*(r+t)
            V_cap1 = (pi/6)*H*(3*(r+t)**2 + H**2)
            A = r
            H = ld_ratio*r
            V_cap2 = (pi/6)*H*(3*A**2 + H**2)
            endcaps_mass = 2*(V_cap1 - V_cap2)*material_density
        barrel_mass = 0.
        if ld_ratio > 1: # capsule shape
            l_tank = ld_ratio*2*r
            l_barrel = l_tank - (2*r)
            barrel_mass = pi*l_barrel*((r+t*2)**2-r**2)*material_density
        bare_tank_mass = endcaps_mass + barrel_mass
        dry_mass += bare_tank_mass

        # weld lands mass
        if ld_ratio == 1:
            weld_lands_mass = 0. # assume monolithic, no welds
        else:
            weld_width = 0.1 # m
            weld_lands_mass = 2*pi*r*t*weld_width*material_density*2
        dry_mass += weld_lands_mass

        # inlet and outlet flange mass
        let_radius = 0.2*r # m
        flange_width = .0508 # m(2 in.)
        flange_height = 0.00635 # m(0.25 in)
        inlet_outlet_mass = (pi*((let_radius+flange_width)**2-let_radius**2)*
                             flange_height*material_density*2)
        dry_mass += inlet_outlet_mass

        # structural attachment points mass
        structural_attach_mass = 0.02 * dry_mass
        dry_mass += structural_attach_mass

        # separation device mass
        if separator_type.lower() == 'pmd':
            separator_mass = 0.2 * dry_mass
        elif separator_type.lower() == 'ped':
            separator_mass = 0.3 * dry_mass
        elif not separator_type:
            separator_mass = 0.
        dry_mass += separator_mass

    return dry_mass, r+t
```

## I.6 Thermal SubElement Model

```
# -*- coding: utf-8 -*-
"""
Description:
    A DYREQT Thermal subelement for Douglas Trent's PhD

Written by:
    Douglas J. Trent
    NASA Marshall Space Flight Center
    Advanced Concept Office
    douglas.trent@nasa.gov

Created: 04/06/2017
Revised: 07/30/2017
"""
# import DYREQT Subelement base class
from SubElements import SubElement
from Constants import Is, SB
# import other modules
import FluidsDef as fluids
from numpy import zeros, pi, log, cos, arcsin

# create the structures subelement
class ThermalPhD(SubElement):
    """Estimates the mass of a thermal control subsystem. Many of the scaling
    equations and mass estimates are derived from references used in development
    of the CryoSim tool by Steve Sutherlin of NASA MSFC and Wesley Johnson of
    NASA KSC, as well as Human Spaceflight Mission Analysis and Design by W. Larson
    and Space Vehicle Design, 2nd ed. by Michael D. Griffin and James R. French

    Input Params
    ------------
    mli_layers_mps : int (20)
        The number of layers in the MLI blankets for the main propulsion system
        tanks.

    mli_layers_rcs : int (20)
        The number of layers in the MLI blankets for the reaction control
        system tanks.

    active_cooling_mps : bool(False)
        If True, include and active cooling system to reduce propellant boil
        off to zero for the main propulsion system.

    active_cooling_mps : bool(False)
        If True, include and active cooling system to reduce propellant boil
        off to zero for the reaction control system.

    radiator_density : float (4.5 kg/m**2)
        The aerial density of the radiator (kg/m^2)

    external_tanks : bool (True)
        If True, propellant tanks are assumed external to the main element
        structure and are directly affected by external radiation sources such
        as the Sun and/or orbited bodies. MLI mass and energy leak will be
        calculated based on tank geometry. If False, MLI mass and energy leak
        are calculated based on assumed element geometry.

    hi_efficiency_radiators : bool (False)
        If True, uses hi-efficiency radiators with high emissivity and high
        fin efficiency to radiate heat at a greater rate for an equivalent
        radiator area.

    ops_distance : float (1.0 AU)
        The solar distance from the sun of the worst operational environment (AU)

    deep_space : bool (False)
        If True, ignore radiation affect near an orbited body.

    orbit_alt : float (1000. km)
        The orbit altitude of the element above the body (km). If not orbiting
        a body, set albedo to zero to ignore body reflections.

    r_body : float (6371. km)
        The radius of the orbited body (km). If not orbiting a body, set albedo
        to zero to ignore body reflections.

    T_body : float (290. K)
        The average temperature of the orbited body (K) for radiation
        calculations.

    albedo : float (0.3)
        The bond albedo of the orbited body. If not orbiting a body, set this
        value to zero and any value for orbit_alt and orbit_radius.

    Inherited Params
    ----------------
    max_propellant_mass_mps : float
        The mass of propellant for the main propulsion system (kg)

    max_propellant_mass_rcs : float
        The mass of propellant for the reaction control system (kg)

    propellants : str
        The propellants of the propulsion system, separated by a forward slash (/)

    mixture_ratio : float
        The mass mixture ratio of propellants for the propulsion system

    heat_loads : array
        The amount of heat generated by other subsystems to be dissipated by
        the radiators (W)

    num_fuel_tanks : int
        The number of fuel tanks

    diameter_fuel_tanks : float
        The diameter of the fuel tanks (m)

    length_fuel_tanks : float
        The length of the fuel tanks (m)

    pressure_fuel_tanks : float
        The pressure of the fuel tank (MPa)

    num_ox_tanks : int
        The number of oxidizer tanks

    diameter_ox_tanks : float
        The diameter of the oxidizer tanks (m)

    length_ox_tanks : float
        The length of the oxidizer tanks (m)

    pressure_ox_tanks : float
        The pressure of the oxidizer tank (MPa)

    Outputs
    -------
```

```
power_req : array
    The power required for the thermal subelement (W)

inert_mass : float
    The inert mass of the subelement (kg)
"""

def __init__(self,**kwargs):
    super().__init__(**kwargs)
    # user model inputs
    self.add_param(self.base_name+'_mli_layers_mps', val=int(20))
    self.add_param(self.base_name+'_mli_layers_rcs', val=int(20))
    self.add_param(self.base_name+'_active_cooling_mps', val=False)
    self.add_param(self.base_name+'_active_cooling_rcs', val=False)
    self.add_param(self.base_name+'_radiator_density', val=float(4.5), units='kg/m**2')
    self.add_param(self.base_name+'_external_tanks', val=True)
    self.add_param(self.base_name+'_hi_efficiency_radiators', val=False)
    self.add_param(self.base_name+'_ops_distance', val=float(1), units='AU') # will eventually be
    ↪  inherited from DYREQT mission
    self.add_param(self.base_name+'_deep_space', val=False) # will eventually be inherited from
    ↪  DYREQT mission
    self.add_param(self.base_name+'_orbit_alt', val=float(1000), units='km') # will eventually be
    ↪  inherited from DYREQT mission
    self.add_param(self.base_name+'_r_body', val=float(6371), units='km') # will eventually be
    ↪  inherited from DYREQT mission
    self.add_param(self.base_name+'_T_body', val=float(290), units='K') # will eventually be
    ↪  inherited from DYREQT mission
    self.add_param(self.base_name+'_albedo', val=float(0.3)) # will eventually be inherited from
    ↪  DYREQT mission
    # parameters from the element (DYREQT internal or from other subelements)
    self.add_param(self.element.base_name+'_max_propellant_mass_mps', val=float(0), units='kg')
    self.add_param(self.element.base_name+'_propellants_mps', val=str())
    self.add_param(self.element.base_name+'_mixture_ratio_mps', val=float(1))
    self.add_param(self.element.base_name+'_num_fuel_tanks_mps', val=int(1))
    self.add_param(self.element.base_name+'_diameter_fuel_tanks_mps', val=float(0), units='m')
    self.add_param(self.element.base_name+'_length_fuel_tanks_mps', val=float(0), units='m')
    self.add_param(self.element.base_name+'_pressure_fuel_tanks_mps', val=float(0.3), units='MPa')
    self.add_param(self.element.base_name+'_num_ox_tanks_mps', val=int(1))
    self.add_param(self.element.base_name+'_diameter_ox_tanks_mps', val=float(0), units='m')
    self.add_param(self.element.base_name+'_length_ox_tanks_mps', val=float(0), units='m')
    self.add_param(self.element.base_name+'_pressure_ox_tanks_mps', val=float(0.3), units='MPa')
    self.add_param(self.element.base_name+'_max_propellant_mass_rcs', val=float(0), units='kg')
    self.add_param(self.element.base_name+'_propellants_rcs', val=str())
    self.add_param(self.element.base_name+'_mixture_ratio_rcs', val=float(1))
    self.add_param(self.element.base_name+'_num_fuel_tanks_rcs', val=int(0))
    self.add_param(self.element.base_name+'_diameter_fuel_tanks_rcs', val=float(0), units='m')
    self.add_param(self.element.base_name+'_length_fuel_tanks_rcs', val=float(0), units='m')
    self.add_param(self.element.base_name+'_pressure_fuel_tanks_rcs', val=float(0.3), units='MPa')
    self.add_param(self.element.base_name+'_num_ox_tanks_rcs', val=int(0))
    self.add_param(self.element.base_name+'_diameter_ox_tanks_rcs', val=float(0), units='m')
    self.add_param(self.element.base_name+'_length_ox_tanks_rcs', val=float(0), units='m')
    self.add_param(self.element.base_name+'_pressure_ox_tanks_rcs', val=float(0.3), units='MPa')
    # outputs inherited by the element (for use by other subelements or DYREQT)
    self.add_output(self.element.base_name+'_boiloff_rate_mps',
    ↪  val=zeros(self.element.num_events), units='1/d')
    self.add_output(self.element.base_name+'_boiloff_rate_rcs',
    ↪  val=zeros(self.element.num_events), units='1/d')
    # outputs used by the parent element
    self.add_output(self.base_name+'_power_req', val=float(0), units='W')
    self.add_output(self.base_name+'_inert_mass', val=float(0), units='kg')
    # heat requirement from all other subelements in the parent element
    for subnum in range(self.element.num_subelements):
        if subnum != self.subelement_num:
            self.add_param('element{0}sub{1}_heat_load'.format(self.element.element_num,subnum),
            ↪  val=float(0), units='W')

def pre_setup(self, problem):
    pass
```

```
def post_setup(self, problem):
    pass

def solve_nonlinear(self, params, unknowns, resids):
    # convert constants to proper units
    Is.convert_to_unit('W/m**2')
    SB.convert_to_unit('W/m**2/K**4')

    # gather params
    active_cooling_mps = params[self.base_name+'_active_cooling_mps']
    active_cooling_rcs = params[self.base_name+'_active_cooling_rcs']
    mli_layers_mps = params[self.base_name+'_mli_layers_mps']
    mli_layers_rcs = params[self.base_name+'_mli_layers_rcs']
    radiator_density = params[self.base_name+'_radiator_density']
    external_tanks = params[self.base_name+'_external_tanks']
    hieff_rad = params[self.base_name+'_hi_efficiency_radiators']
    ops_distance = params[self.base_name+'_ops_distance']
    deep_space = params[self.base_name+'_deep_space']
    orbit_alt = params[self.base_name+'_orbit_alt']
    r_body = params[self.base_name+'_r_body']
    T_body = params[self.base_name+'_T_body']
    albedo = params[self.base_name+'_albedo']
    propellants_mps = params[self.element.base_name+'_propellants_mps']
    mr_mps = params[self.element.base_name+'_mixture_ratio_mps']
    num_fuel_tanks_mps = params[self.element.base_name+'_num_fuel_tanks_mps']
    diameter_fuel_tanks_mps = params[self.element.base_name+'_diameter_fuel_tanks_mps']
    length_fuel_tanks_mps = params[self.element.base_name+'_length_fuel_tanks_mps']
    pressure_fuel_tanks_mps = params[self.element.base_name+'_pressure_fuel_tanks_mps']
    num_ox_tanks_mps = params[self.element.base_name+'_num_ox_tanks_mps']
    diameter_ox_tanks_mps = params[self.element.base_name+'_diameter_ox_tanks_mps']
    length_ox_tanks_mps = params[self.element.base_name+'_length_ox_tanks_mps']
    pressure_ox_tanks_mps = params[self.element.base_name+'_pressure_ox_tanks_mps']
    propellants_rcs = params[self.element.base_name+'_propellants_rcs']
    mr_rcs = params[self.element.base_name+'_mixture_ratio_rcs']
    num_fuel_tanks_rcs = params[self.element.base_name+'_num_fuel_tanks_rcs']
    diameter_fuel_tanks_rcs = params[self.element.base_name+'_diameter_fuel_tanks_rcs']
    length_fuel_tanks_rcs = params[self.element.base_name+'_length_fuel_tanks_rcs']
    pressure_fuel_tanks_rcs = params[self.element.base_name+'_pressure_fuel_tanks_rcs']
    num_ox_tanks_rcs = params[self.element.base_name+'_num_ox_tanks_rcs']
    diameter_ox_tanks_rcs = params[self.element.base_name+'_diameter_ox_tanks_rcs']
    length_ox_tanks_rcs = params[self.element.base_name+'_length_ox_tanks_rcs']
    pressure_ox_tanks_rcs = params[self.element.base_name+'_pressure_ox_tanks_rcs']

    heat_load = 0.
    for subnum in range(self.element.num_subelements):
        if subnum != self.subelement_num:
            heat_load +=
            ↪  params['element{0}sub{1}_heat_load'.format(self.element.element_num,subnum)]

    tank_diameters = [diameter_fuel_tanks_mps,
                      diameter_ox_tanks_mps,
                      diameter_fuel_tanks_rcs,
                      diameter_ox_tanks_rcs]

    tank_lengths = [length_fuel_tanks_mps,
                    length_ox_tanks_mps,
                    length_fuel_tanks_rcs,
                    length_ox_tanks_rcs]

    num_tanks = [num_fuel_tanks_mps,
                 num_ox_tanks_mps,
                 num_fuel_tanks_rcs,
                 num_ox_tanks_rcs]

    d_tanks = 0.
    l_tanks = 0.
    for i in range(0,4):
```

```
    d_tanks += tank_diameters[i] * num_tanks[i]/sum(num_tanks)
    l_tanks += tank_lengths[i] * num_tanks[i]/sum(num_tanks)

if sum(num_tanks[0:2]) == 2: # assume two stacked stanks
    d_sc = d_tanks
    l_sc = l_tanks
else: # assume disk shape
    # diameter of a circle which fits the 120% diameter tanks
    d_sc = (1.2*d_tanks)*(1.1655*log(sum(num_tanks))+0.9571)
    l_sc = 1.2*l_tanks

Asc = max(d_sc * l_sc,pi*(d_sc/2)**2) # spacecraft cross section area, assumes worst case
↪  orientation

solar_flux = Is.value * (1/ops_distance**2) # W/m^2 # inverse square law relation

Tsc = 280. # K, assumed spacecraft temperature

if not deep_space:
    sub_angle = arcsin(r_body/(r_body + orbit_alt)) # the subtended angle of the body
    F_sb = 2*pi*(1-cos(sub_angle)) / (4*pi) # view factor of orbited body to spacecraft
else:
    F_sb = 0.

Qthermal = 0.
mass_passive = 0.
mass_active = 0.
power_passive = 0.
power_active = 0.
boiloff_rate_mps = 0.
boiloff_rate_rcs = 0.
sc_flag = False

for sysname in ['mps','rcs']:

    if sysname == 'mps':
        propellants = propellants_mps
        mixture_ratio = mr_mps
        geom_fuel_tanks = [diameter_fuel_tanks_mps,length_fuel_tanks_mps]
        geom_ox_tanks = [diameter_ox_tanks_mps,length_ox_tanks_mps]
        num_fuel_tanks = num_fuel_tanks_mps
        num_ox_tanks = num_ox_tanks_mps
        tank_pressure = [pressure_fuel_tanks_mps,pressure_ox_tanks_mps]
        active_cooling = active_cooling_mps
        mli_layers = mli_layers_mps
    elif sysname == 'rcs':
        propellants = propellants_rcs
        mixture_ratio = mr_rcs
        geom_fuel_tanks = [diameter_fuel_tanks_rcs,length_fuel_tanks_rcs]
        geom_ox_tanks = [diameter_ox_tanks_rcs,length_ox_tanks_rcs]
        num_fuel_tanks = num_fuel_tanks_rcs
        num_ox_tanks = num_ox_tanks_rcs
        tank_pressure = [pressure_fuel_tanks_rcs,pressure_ox_tanks_rcs]
        active_cooling = active_cooling_rcs
        mli_layers = mli_layers_rcs

    if propellants:
        props_list = propellants.replace(' ','').lower().split('/')

        ox = None
        fuel = None
        if len(props_list) >1:
            ox = props_list[0]
            fuel = props_list[1]
        else:
            if props_list[0]:
                fuel = props_list[0]
```

```
# perform some input checks
prop_types = []
for i,propellant in enumerate([fuel,ox]):
    if propellant:
        prop_types.append(None)
        if propellant == 'solid':
            prop_types[i] = 'solid'
        else:
            res,fluid_def = fluids.check_def(propellant)
            if not res:
                msg = ('"{0}" is not a defined propellant'.format(propellant))
                raise Exception(msg)
            else:
                prop_types[i] = fluid_def['type']

# determine the mass of fuel and oxidizer
if fuel and ox:
    pass
else:
    if prop_types[0] == 'solid':
        continue # skip sizing passive/active for this system and move to the next

##########################################################################
# passive TCS sizing
##########################################################################

Qmli_fuel = 0.
Qmli_ox = 0.
SA_mli = 0.
SA_tanks = 0.
n = mli_layers

if external_tanks:
    Qmli = [0.,0.]
    Tc_props = [None,None]
    tank_geoms = [geom_fuel_tanks,geom_ox_tanks]
    num_tanks = [num_fuel_tanks,num_ox_tanks]
    for idx,prop in enumerate([fuel,ox]):
        if prop:
            geom_tanks = tank_geoms[idx]
            num_prop_tanks = num_tanks[idx]
            if num_prop_tanks > 0:
                # assume tanks are either spheres or cylinders only
                if geom_tanks[1] > geom_tanks[0]:
                    l_barrel = geom_tanks[1] - geom_tanks[0]
                    sa_prop = 4*pi*(geom_tanks[0]/2)**2 +
                    ↪  2*pi*geom_tanks[0]/2*l_barrel
                    cs_prop = 2*pi*geom_tanks[0]/2 + (geom_tanks[0]*l_barrel)
                else:
                    sa_prop = 4*pi*(geom_tanks[0]/2)**2
                    cs_prop = 2*pi*geom_tanks[0]/2
                sa_prop = sa_prop * num_prop_tanks

                SA_tanks += sa_prop
                SA_mli += sa_prop

                n = mli_layers # number of mli layers

                alpha_mli = 0.1 # assumed based on vapor deposited aluminum sheets in
                ↪  MLI
                emis_mli = 0.34*(1/(1+n))

                # radiation equations from Space Vehicle Design, 2nd ed. by Michael D.
                ↪  Griffin
                # and James R. French, sec. 9.5.2, p. 463-465
                Qbref = albedo*alpha_mli*F_sb*cs_prop*solar_flux # orbited body
                ↪  reflected heat
                Qbrad = SB.value*cs_prop*F_sb*(T_body**4 - Tsc**4) # orbited body
                ↪  radiated heat
                Qsun  = alpha_mli*cs_prop*solar_flux # heat absorbed by the spacecraft
                ↪  from the sun

                Th = ((Qbref + Qbrad + Qsun) / (SB.value*emis_mli*sa_prop))**(0.25) #
                ↪  hot side mli temp

                if prop_types[idx] == 'cryogenic':
                    Tc = fluids.tvap(prop,tank_pressure[idx])
                    Tc_props[idx] = Tc
                else:
                    Tc_props[idx] = Tsc # K

                if Th < Tc:
                    Th = Tc+1

                # W, The Lockheed Equation for estimating heat penetration through an
                ↪  MLI blanket
                try:
                    Qmli[idx] = sa_prop * (2.4e-4*(.017 + 7e-6*(800 - (Th-Tc)/2) +
                    ↪  .0228*log((Th-Tc)/2))*15**2.63*(Th-Tc) +
                    ↪  4.944e-10*emis_mli*(Th**4.67 - Tc**4.67)) / n
                except Exception:
                    pass

        Qmli_fuel = Qmli[0]
        Qmli_ox = Qmli[1]

        Tc_fuel = Tc_props[0]
        Tc_ox = Tc_props[1]
    else:
        tank_geoms = [geom_fuel_tanks,geom_ox_tanks]
        num_tanks = [num_fuel_tanks,num_ox_tanks]
        for idx,prop in enumerate([fuel,ox]):
            if prop:
                geom_tanks = tank_geoms[idx]
                num_prop_tanks = num_tanks[idx]
                if num_prop_tanks > 0:
                    # assume tanks are either spheres or cylinders only
                    if geom_tanks[1] > geom_tanks[0]:
                        l_barrel = geom_tanks[1] - geom_tanks[0]
                        sa_prop = 4*pi*(geom_tanks[0]/2)**2 +
                        ↪  2*pi*geom_tanks[0]/2*l_barrel
                        cs_prop = 2*pi*geom_tanks[0]/2 + (geom_tanks[0]*l_barrel)
                    else:
                        sa_prop = 4*pi*(geom_tanks[0]/2)**2
                        cs_prop = 2*pi*geom_tanks[0]/2
                    sa_prop = sa_prop * num_prop_tanks

                    SA_tanks += sa_prop

        SA_mli = 2*pi*(d_sc/2)**2 + (2*pi*d_sc/2)*l_sc

        if any([True for prop_type in prop_types if prop_type == 'cryogenic']):
            CS = Asc

            n = mli_layers # number of mli layers

            alpha_mli = 0.1 # assumed based on vapor deposited aluminum sheets in MLI
            emis_mli = 0.34*(1/(1+n))

            # radiation equations from Space Vehicle Design, 2nd ed. by Michael D. Griffin
            # and James R. French, sec. 9.5.2, p. 463-465
            Qbref = albedo*alpha_mli*F_sb*CS*solar_flux # orbited body reflected heat
            Qbrad = SB.value*CS*F_sb*(T_body**4 - Tsc**4) # orbited body radiated heat
            Qsun  = alpha_mli*CS*solar_flux # heat absorbed by the spacecraft from the sun
            Tc = Tsc # K

            if SA_mli:
                Th = ((Qbref + Qbrad + Qsun) / (SB.value*emis_mli*SA_mli))**(0.25) # hot
                ↪  side mli temp
                if Th < Tc:
                    Th = Tc+1
            else:
                Th = Tsc+1

            # W, The Lockheed Equation for estimating heat penetration through an MLI
            ↪  blanket
            try:
                Qmli = SA_mli * (2.4e-4*(.017 + 7e-6*(800 - (Th-Tc)/2) +
                    ↪  .0228*log((Th-Tc)/2))*15**2.63*(Th-Tc) + 4.944e-10*emis_mli*(Th**4.67
                    ↪  - Tc**4.67)) / n
            except Exception:
                Qmli = 0.

        if fuel:
            if prop_types[0] == 'cryogenic':
                Qmli_fuel = Qmli
                Tc_fuel = fluids.tvap(fuel,tank_pressure[idx])
        if ox:
            if prop_types[1] == 'cryogenic':
                Qmli_ox = Qmli
                Tc_ox = fluids.tvap(ox,tank_pressure[idx])

        if sc_flag:
            SA_mli = 0.
        else:
            sc_flag = True

    mass_mli = 1.1 * (SA_mli * 0.018 * n) # 0.018 kg/m^2/layer, 1.1 for hardware mass

    mass_lad = SA_tanks*0.57 # 0.57 kg/square meter of LAD weight for vanes,
    ↪  Debreceini(1997), Tam(1998)

    # based on Haberbusch, et al., "Reduced-Gravity Cryo-Tracker System",
    # AIAA, January 2009.
    TankLongestDim = max(geom_fuel_tanks + geom_ox_tanks)
    PenetrationMass = 10. # Approximate mass of penetration assembly, kg
    AvionicsMass = 10. # Approximate mass of associated avionics, kg
    VariableMass = 1.*TankLongestDim # Mass per unit tank dimension for cabling, etc, kg
    mass_gauging = PenetrationMass+AvionicsMass+VariableMass # Total mass gauging system
    ↪  mass, kg

    NominalPower = 100. # Nominal power for primary & backup gauge systems, W
    VariablePower = 1.*TankLongestDim # Power per unit tank dimension for losses, etc, W
    power_gauging = NominalPower+VariablePower # Total gauge system power, W

    mass_passive += mass_mli + mass_lad + mass_gauging
    power_passive += power_gauging

    # calculate passive boiloff rates
    boiloff_rate_fuel = 0.
    boiloff_rate_ox = 0.

    if fuel:
        if prop_types[0] == 'cryogenic':
            Hvap_fuel = fluids.get_property(fuel,'Hvap') * 1000. # J/kg, propellant
            ↪  dependent
            boiloff_rate_fuel = Qmli_fuel * 86400 / Hvap_fuel # kg/d

    if ox:
        if prop_types[1] == 'cryogenic':
            Hvap_ox = fluids.get_property(ox,'Hvap') * 1000. # J/kg, propellant dependent
            boiloff_rate_ox = Qmli_ox * 86400 / Hvap_ox # kg/d
        boiloff_rate_prop = (1 / (mixture_ratio + 1)) * boiloff_rate_fuel + (mixture_ratio /
        ↪  (mixture_ratio + 1)) * boiloff_rate_ox # kg/d

        ###########################################################################
        # active TCS sizing
        ###########################################################################

        if active_cooling:

            boiloff_rate_prop = 0.

            Qlift_fuel = Qmli_fuel
            Qlift_ox = Qmli_ox

            # the following equations for scaling power and mass of ATC are from
            # Preliminary Study of Lunar Lander Descent Stage Active Thermal Control Systems
            # by J. R. Feller of NASA Ames Research Center, 21 March 2011 (unpublished)

            power_cc = 0.
            mass_cc = 0.
            Qlift = 0.

            if Qlift_fuel > 0. and num_fuel_tanks:

                power_cc_fuel = Qlift_fuel * 10. * (Tc_fuel**0.61 / 90.**0.6)**-2.066
                mass_cc_fuel = 1.5 * 0.0711 * power_cc_fuel**0.905

                power_cc += power_cc_fuel
                mass_cc += mass_cc_fuel
                Qlift += Qlift_fuel

            if Qlift_ox > 0. and num_ox_tanks:

                power_cc_ox = Qlift_ox * 10. * (Tc_ox**0.61 / 90.**0.6)**-2.066
                mass_cc_ox = 1.5 * 0.0711 * power_cc_ox**0.905

                power_cc += power_cc_ox
                mass_cc += mass_cc_ox
                Qlift += Qlift_ox

            mass_controller = 1.5 * 0.01 * power_cc

            power_circ = 0.72 * Qlift

            mass_circ = 0.042 * power_circ

            mass_shield = 1.5 * SA_tanks * 0.42

            mass_tubing = 1.5 * SA_tanks * 0.018

            mass_active += mass_cc + mass_controller + mass_circ + mass_shield + mass_tubing
            power_active += power_cc + power_circ
            Qthermal += Qlift

        if sysname == 'mps':
            boiloff_rate_mps = boiloff_rate_prop
        elif sysname == 'rcs':
            boiloff_rate_rcs = boiloff_rate_prop

###########################################################################
# spacecraft energy balancer and radiator sizing
###########################################################################

alpha_sc = 0.1 # average absorptivity of the spacecraft, Assumes single
# layer mli over exposed surfaces

# radiation equations from Space Vehicle Design, 2nd ed. by Michael D. Griffin
```

```
        # and James R. French, sec. 9.5.2, p. 463-465
        Qbref = albedo*alpha_sc*F_sb*Asc*solar_flux # orbited body reflected heat
        Qbrad = SB.value*Asc*F_sb*(T_body**4 - Tsc**4) # orbited body radiated heat

        Qin = heat_load + Qthermal # internal heat loads generated by spacecraft
        Qss = alpha_sc*Asc*solar_flux # heat absorbed by the spacecraft from the sun
        Qsb = Qbref + Qbrad # heat absorbed by the spacecraft from the orbited body

        Q = Qin + Qss + Qsb # total heat input that must be radiated by the radiator

        # constants assumed from typical values given in Human Spaceflight
        # Mission Analysis and Design, p. 521
        # can bring these parameters in as technology controls for high
        # efficiency radiators
        if hieff_rad:
            emis_rad = 0.9
            fin_efficiency = 0.95
        else:
            emis_rad = 0.8
            fin_efficiency = 0.85
        Tspace = 2.7 # K
        Trad = 250.0 # K assumed radiator surface temperature

        # Eq. 16-4 from Human Spaceflight Mission Analysis and Design, p. 521
        # Assumes radiators have very low absorptivity, resulting in negligible
        # solar absorption.
        Arad = Q / (SB.value * emis_rad * fin_efficiency * (Trad**4 - Tspace**4))

        mass_rad = Arad * radiator_density

        inert_mass = mass_passive + mass_active + mass_rad

        power_req = power_passive + power_active

        # assign unknowns
        unknowns[self.element.base_name+'_boiloff_rate_mps'].fill(boiloff_rate_mps)
        unknowns[self.element.base_name+'_boiloff_rate_rcs'].fill(boiloff_rate_rcs)
        unknowns[self.base_name+'_power_req'] = power_req
        unknowns[self.base_name+'_inert_mass'] = inert_mass
```

## I.7 Burn Event Model

```
class Burn(Event):
    """An Event subclass to account for impulsive delta-V maneuvers. Parallel
    burning elements result in an average Isp based on the performance of all
    active elements.

    Input Params
    ------------
    dv : float(1 m/s)
        The impulsive delta-V of the event.
    system : str('MPS','RCS')
        The propulsion system of the active element(s) to use for the burn
    fpr : float(0.0)
        The Flight Performance Reserve (FPR) as a percentage of the event
        delta-V input to be added to the required event delta-V.
    acs_factor : float(0.0)
        Attitude Control System (ACS) factor as a percentage of the event
        delta-V (including FPR) to be added as an attitude control dv. The
        attitude control burn always utilizes the RCS propulsion and is split
        before and after the main event burn. Use the 'acs_split' setting to
        adjust the fraction of the ACS maneuver performed before and after the
        main event burn.
    acs_split : float(50.0)
        Percentage of the ACS maneuver to perform BEFORE the main event burn.
```

```
Args
----
"""

def __init__(self,**kwargs):
    super().__init__(**kwargs)
    self.Setup_Elements(self.element_list)
    self.add_param(self.base_name+'_dv', val = 1.0, units='m/s')
    self.add_param(self.base_name+'_system', val = 'mps')
    self.add_param(self.base_name+'_fpr', val = 0.0)
    self.add_param(self.base_name+'_acs_factor', val = 0.0)
    self.add_param(self.base_name+'_acs_split', val = 50.0)
    self.Build_Equivalent_Stages()
    for idx in range(len(self.element_list)-1):
        self.add_param('opt_var_'+str(idx)+'_dv_'+self.base_name, val = 0.5)
    self.defined_thrust = [] # same shape as element_list
    for segment_num in range(len(self.element_list)):
        self.add_output(self.base_name+'_'+str(segment_num)+'_dt', val=0.0, units='s')
    self.add_output(self.base_name+'_propellant_mass_main', val=0.0, units='kg')
    self.add_output(self.base_name+'_propellant_mass_acs', val=0.0, units='kg')
    self.add_output(self.base_name+'_sized_dv', val=1.0, units='m/s')

def Setup_Elements(self,element_list):
    """Sets up the element. This method is called during initialization of
    the object

    Args
    ----
    element_list : list
        A list of event segment active element lists. Each event segment
        active element list contains integers referencing an element
        index.
    """
    elements_already_mapped = []
    for idx,element in enumerate(element_list):
        base_name = self.base_name+'_'+str(idx)
        for sub_idx,sub_element in enumerate(element):
            target_name = 'element'+str(sub_element)
            if sub_element not in elements_already_mapped:
                self.add_param(target_name+'_isp_mps', val = 1.0, units='s')
                self.add_param(target_name+'_isp_rcs', val = 1.0, units='s')
                self.add_param(target_name+'_mass_flowrate_mps', val = 1.0, units='kg/s')
                self.add_param(target_name+'_mass_flowrate_rcs', val = 1.0, units='kg/s')
                self.add_param(target_name+'_inert_mass', val=ones(self.num_events+1), units='kg',
                ↪  pass_by_obj=True)
                self.add_param(target_name+'_terminal_event', val=(0,0))
                elements_already_mapped.append(sub_element)
            self.add_output(target_name+'_burn_time_mps_'+base_name, val=1.0, units='s')
            self.add_output(target_name+'_burn_time_rcs_'+base_name, val=1.0, units='s')

def Build_Equivalent_Stages(self):
    """Adds outputs to the component to create equivalent stages,
    a representations of multiple burning propulsive stages into a single
    propulsive stage.
    """
    for idx,_ in enumerate(self.element_list):
        self.add_output('equivalent_stage'+'_'+str(self.event_num)+'_'+str(idx)+'_thrust_main',
        ↪  val=1.0, units='N')
        self.add_output('equivalent_stage'+'_'+str(self.event_num)+'_'+str(idx)+'_thrust_acs',
        ↪  val=1.0, units='N')

        ↪  self.add_output('equivalent_stage'+'_'+str(self.event_num)+'_'+str(idx)+'_mass_flowrate_main',
        ↪  val=1.0, units='kg/s')

        ↪  self.add_output('equivalent_stage'+'_'+str(self.event_num)+'_'+str(idx)+'_mass_flowrate_acs',
        ↪  val=1.0, units='kg/s')
```

```
            self.add_output('equivalent_stage'+'_'+str(self.event_num)+'_'+str(idx)+'_isp_main',
            ↪  val=1.0, units='s')
            self.add_output('equivalent_stage'+'_'+str(self.event_num)+'_'+str(idx)+'_isp_acs',
            ↪  val=1.0, units='s')
            self.add_output('equivalent_stage'+'_'+str(self.event_num)+'_'+str(idx)+'_jettison_mass',
            ↪  val=1.0, units='kg')
            self.add_output('equivalent_stage'+'_'+str(self.event_num)+'_'+str(idx)+'_initial_mass',
            ↪  val=1.0, units='kg')
            self.add_output('equivalent_stage'+'_'+str(self.event_num)+'_'+str(idx)+'_final_mass',
            ↪  val=1.0, units='kg')
            self.add_output('equivalent_stage'+'_'+str(self.event_num)+'_'+str(idx)+'_t2w', val=1.0)
            self.add_output('equivalent_stage'+'_'+str(self.event_num)+'_'+str(idx)+'_dv', val=1.0,
            ↪  units='m/s')
            self.add_output('equivalent_stage'+'_'+str(self.event_num)+'_'+str(idx)+'_burn_time_main',
            ↪  val=1.0, units='s')
            self.add_output('equivalent_stage'+'_'+str(self.event_num)+'_'+str(idx)+'_burn_time_acs',
            ↪  val=1.0, units='s')

    def Calculate_Equivalent_Stages(self,params,unknowns):
        """Calculates parameters to define an equivalent stage, such as the
        combined thrust, flow rate, isp, etc. of the equivalent stage from its
        constituent stages.
        """
        G0.convert_to_unit('m/s**2')
        for idx,element in enumerate(self.element_list):
            base_name = 'equivalent_stage'+'_'+str(self.event_num)+'_'+str(idx)
            unknowns[base_name+'_thrust_main'] = 0.0
            unknowns[base_name+'_thrust_acs'] = 0.0
            unknowns[base_name+'_mass_flowrate_main'] = 0.0
            unknowns[base_name+'_mass_flowrate_acs'] = 0.0
            unknowns[base_name+'_isp_main'] = 0.0
            unknowns[base_name+'_isp_acs'] = 0.0
            unknowns[base_name+'_jettison_mass'] = 0.0
            system = params[self.base_name+'_system'].lower()
            if system not in ['mps','rcs']:
                raise Exception('Invalid system setting input. Must be one of ["mps","rcs"]')
            for parallel_element in element:
                target_name = 'element'+str(parallel_element)
                unknowns[base_name+'_thrust_main'] += params[target_name+'_mass_flowrate_'+system] *
                ↪  params[target_name+'_isp_'+system] * G0.value
                unknowns[base_name+'_thrust_acs'] += params[target_name+'_mass_flowrate_rcs'] *
                ↪  params[target_name+'_isp_rcs'] * G0.value
                unknowns[base_name+'_mass_flowrate_main'] +=
                ↪  params[target_name+'_mass_flowrate_'+system]
                unknowns[base_name+'_mass_flowrate_acs'] += params[target_name+'_mass_flowrate_rcs']
                if params[target_name+'_terminal_event'] == (self.event_num,idx):
                    unknowns[base_name+'_jettison_mass'] +=
                    ↪  params[target_name+'_inert_mass'][self.event_num]
            unknowns[base_name+'_isp_main'] = unknowns[base_name+'_thrust_main'] /
            ↪  unknowns[base_name+'_mass_flowrate_main'] / G0.value
            unknowns[base_name+'_isp_acs'] = unknowns[base_name+'_thrust_acs'] /
            ↪  unknowns[base_name+'_mass_flowrate_acs'] / G0.value
#            print('{0},{1}'.format(unknowns[base_name+'_isp_main'],unknowns[base_name+'_isp_acs']))

    def Allocate_DeltaV(self,params,unknowns):
        """Allocate delta-V to the equivalent stages based on the optimization
        variables for delta-V allocation. This allocation can be fixed by
        defining a fixed optimization variable at the architecture level.
        """
        dv_remaining = params[self.base_name+'_dv'] * (1 + params[self.base_name+'_fpr']/100.)
        unknowns[self.base_name+'_sized_dv'] = dv_remaining
        # sometimes, Gauss-Seidel likes to try very small numbers just past
        # the user defined range. In the event the users specifies a lower
        # bound on a dv range as 0.0, this will help stabilize the problem
        # when Gauss-Seidel tries a -0.0001 dv.
        if dv_remaining < 0.:
            dv_remaining = 0.
        ''' setup delta-V assignments '''
        for idx in range(len(self.element_list)-1):
            base_name = 'equivalent_stage'+'_'+str(self.event_num)+'_'+str(idx)
            dv_increment = params['opt_var_'+str(idx)+'_dv_'+self.base_name] * dv_remaining
            unknowns[base_name+'_dv'] = dv_increment
            dv_remaining -= dv_increment
        unknowns['equivalent_stage'+'_'+str(self.event_num)+'_'+str(len(self.element_list)-1)+'_dv'] =
        ↪  dv_remaining

    def Calculate_Burn_Times(self,params,unknowns):
        """Calculates the burn time required of an equivalent stage based on
        a required delta-V via the ideal rocket equation.
        """
        G0.convert_to_unit('m/s**2')
        current_mass = params['vehicle_gross_mass'][self.event_num+1]
        unknowns[self.base_name+'_final_mass'] = current_mass
        for idx in reversed(range(len(self.element_list))):
            base_name = 'equivalent_stage'+'_'+str(self.event_num)+'_'+str(idx)
            acs_dv = unknowns[base_name+'_dv'] * params[self.base_name+'_acs_factor'] / 100.
            current_mass += unknowns[base_name+'_jettison_mass']
            unknowns[base_name+'_final_mass'] = current_mass
            # pre-acs burn
            acs_propellant_mass_0 = current_mass * (exp(acs_dv*params[self.base_name+'_acs_split'] /
            ↪  100. / unknowns[base_name+'_isp_acs'] / G0.value) - 1)
            current_mass += acs_propellant_mass_0
            # main event burn
            main_propellant_mass = current_mass * (exp(unknowns[base_name+'_dv'] /
            ↪  unknowns[base_name+'_isp_main'] / G0.value) - 1)
            current_mass += main_propellant_mass
            # post-acs burn
            acs_propellant_mass_1 = current_mass * (exp(acs_dv * (100. -
                ↪  params[self.base_name+'_acs_split']) / 100. / unknowns[base_name+'_isp_acs'] /
                ↪  G0.value) - 1)
            current_mass += acs_propellant_mass_1
            acs_propellant_mass = acs_propellant_mass_0 + acs_propellant_mass_1
            unknowns[base_name+'_burn_time_main'] = main_propellant_mass /
            ↪  unknowns[base_name+'_mass_flowrate_main']
            unknowns[base_name+'_burn_time_acs'] = acs_propellant_mass /
            ↪  unknowns[base_name+'_mass_flowrate_acs']
            unknowns[self.base_name+'_'+str(idx)+'_dt'] = unknowns[base_name+'_burn_time_main']
            unknowns[base_name+'_initial_mass'] = current_mass
            unknowns[base_name+'_t2w'] = unknowns[base_name+'_thrust_main'] / current_mass / G0.value
        unknowns[self.base_name+'_initial_mass'] = current_mass
        unknowns[self.base_name+'_propellant_mass_main'] = main_propellant_mass
        unknowns[self.base_name+'_propellant_mass_acs'] = acs_propellant_mass

    def Assign_Burn_Times(self,params,unknowns):
        """Assigns the calculated burn time of the equivalent stages to their
        associated physical stages for sizing.
        """
        for idx,element in enumerate(self.element_list):
            base_name = self.base_name+'_'+str(idx)
            for parallel_element in element:
                target_name = 'element'+str(parallel_element)
                unknowns[target_name+'_burn_time_mps_'+base_name] = 0.
                unknowns[target_name+'_burn_time_rcs_'+base_name] = 0.
                system = params[self.base_name+'_system'].lower()
                unknowns[target_name+'_burn_time_'+system+'_'+base_name] =
                ↪  unknowns['equivalent_stage'+'_'+str(self.event_num)+'_'+str(idx)+'_burn_time_main']
                unknowns[target_name+'_burn_time_rcs_'+base_name] +=
                ↪  unknowns['equivalent_stage'+'_'+str(self.event_num)+'_'+str(idx)+'_burn_time_acs']

    def solve_nonlinear(self,params,unknowns,resids):
        """Allocate event delta-V, calculate equivalent stage burn times and
        assign those burn times to the associated physical stages for sizing.
        """
        self.Calculate_Equivalent_Stages(params,unknowns)
        self.Allocate_DeltaV(params,unknowns)
```

```
        self.Calculate_Burn_Times(params,unknowns)
        self.Assign_Burn_Times(params,unknowns)
```

## I.8 Connect Event Model

```
class Connect(Event):
    """An Event subclass to account for full element mass additions to the vehicle
    during the mission.

    Input Params
    ------------

    Args
    ----
    """

    def __init__(self,**kwargs):
        super().__init__(**kwargs)
        self.setup_elements(self.element_list)

    def setup_elements(self, element_list):
        """Sets up the element. This method is called during initialization of
        the object

        Args
        ----
        element_list : list
            A list of event segment active element lists. Each event segment
            active element list is a list of integers referencing an element
            index.
        """
        pass

    def solve_nonlinear(self, params, unknowns, resids):
        # calculate initial and final mass
        initial_mass = params['vehicle_gross_mass'][self.event_num]
        unknowns[self.base_name+'_initial_mass'] = initial_mass
        unknowns[self.base_name+'_final_mass'] = params['vehicle_gross_mass'][self.event_num+1]
        for segment_active_elements in self.element_list:
            for element_num in segment_active_elements:
                pass
```

## I.9 Drop Event Model

```
class Drop(Event):
    """An Event subclass to account for full element mass drops off the vehicle
    during the mission.

    Input Params
    ------------

    Args
    ----
    """

    def __init__(self,**kwargs):
        super().__init__(**kwargs)
        self.setup_elements(self.element_list)

    def setup_elements(self, element_list):
        """Sets up the element. This method is called during initialization of
        the object

        Args
        ----
        element_list : list
            A list of event segment active element lists. Each event segment
            active element list is a list of integers referencing an element
            index.
        """
        pass

    def solve_nonlinear(self, params, unknowns, resids):
        # calculate initial and final mass
        initial_mass = params['vehicle_gross_mass'][self.event_num]
        unknowns[self.base_name+'_initial_mass'] = initial_mass
        unknowns[self.base_name+'_final_mass'] = params['vehicle_gross_mass'][self.event_num+1]
        for segment_active_elements in self.element_list:
            for element_num in segment_active_elements:
                pass
```

## I.10 Idle Event Model

```
class Idle(Event):
    """An Event subclass to account for time-based mission effect such as
    propellant boil off, crew consumables, etc.

    Input Params
    ------------
    dt: float(1 day)
        The idle duration for the event.

    Args
    ----
    """

    def __init__(self,**kwargs):
        super().__init__(**kwargs)
        self.Setup_Elements(self.element_list)

        self.add_param(self.base_name+'_dt', val=0., units='d')

    def Setup_Elements(self,element_list):
        """Sets up the element. This method is called during initialization of
        the object

        Args
        ----
        element_list : list
            A list of event segment active element lists. Each event segment
            active element list is a list of integers referencing an element
            index.
        """

        for segment_num,active_elements in enumerate(element_list):
            base_name = self.base_name+'_'+str(segment_num)
            for element_num in active_elements:
                target_name = 'element'+str(element_num)
                self.add_output(target_name+'_idle_time_'+base_name, val=0.0, units='d')

    def solve_nonlinear(self,params,unknowns,resids):
        # calculate initial and final event masses
        unknowns[self.base_name+'_initial_mass'] = params['vehicle_gross_mass'][self.event_num]
        unknowns[self.base_name+'_final_mass'] = params['vehicle_gross_mass'][self.event_num+1]
        # set idle times to be fed to respective elements
        for segment_num,active_elements in enumerate(self.element_list):
            base_name = self.base_name+'_'+str(segment_num)
            for element_num in active_elements:
                target_name = 'element'+str(element_num)
                unknowns[target_name+'_idle_time_'+base_name] = params[self.base_name+'_dt']
```

## I.11 MassDelta Event Model

```
class MassDelta(Event):
    """Discrete mass changes to the vehicle during the mission. These discrete
    changes may be due to loading or unloading mass
    (propellant, consumables, etc.). Maybe also could be used to set certain
    vehicle parameters mid flight such as number of engines
    (dumping engine mass) or other operational possibilities.

    Input Params
    ------------
    dm : float(0 kg)
        The mass delta of the active elements of this event
    sub : list
        The subelement number(s) of the subelement inert mass to remove
    mass_type : str('inert','propellant')
        The type of mass to add/subtract from the active elements
    top_off : bool(False,True)
        Logical control flag which allows the solver to determine the amount
        of propellant to be added to a Stage element to allow it to perform
        its required mission segments without any excess propellant. If this
        flag is true, any value provided in 'dm' will be ignored. This flag has
        an effect on Stage elements types.

    Args
    ----
    """

    def __init__(self,**kwargs):
        self.sub = []
        if 'sub' in kwargs.keys():
            self.sub = kwargs.pop('sub')
        super().__init__(**kwargs)
        self.setup_elements(self.element_list)

        self.add_param(self.base_name+'_dm', val=0., units='kg')
        self.add_param(self.base_name+'_sub', val=-1)
        self.add_param(self.base_name+'_mass_type', val='inert')
        self.add_param(self.base_name+'_top_off', val=False)

    def setup_elements(self, element_list):
        """Sets up the element. This method is called during initialization of
        the object

        Args
        ----
        element_list : list
            A list of event segment active element lists. Each event segment
            active element list is a list of integers referencing an element
            index.
        sub : int
            The subelement index of the subelement inert mass to remove
        """
        for segment_num,active_elements in enumerate(element_list):
            base_name = self.base_name+'_'+str(segment_num)
            for element_num in active_elements:
                target_name = 'element'+str(element_num)
                self.add_output(target_name+'_propellant_mass_delta_'+base_name, val=0.0, units='kg')
                self.add_output(target_name+'_inert_mass_delta_'+base_name, val=0.0, units='kg')
                self.add_output(target_name+'_top_off_'+base_name, val=False, pass_by_obj=True)
                for subidx in self.sub:
                    self.add_param(target_name+'sub'+str(subidx)+'_inert_mass', val=0.0, units='kg')

    def solve_nonlinear(self,params,unknowns,resids):
        # calculate initial and final event masses
        unknowns[self.base_name+'_initial_mass'] = params['vehicle_gross_mass'][self.event_num]
        final_mass = params['vehicle_gross_mass'][self.event_num+1]
        unknowns[self.base_name+'_final_mass'] = final_mass
        # set inert/prop mass deltas to be fed to respective elements
        for segment_num,active_elements in enumerate(self.element_list):
            base_name = self.base_name+'_'+str(segment_num)
            for element_num in active_elements:
                target_name = 'element'+str(element_num)
                mass_type = params[self.base_name+'_mass_type']
                unknowns[target_name+'_top_off_'+base_name] = params[self.base_name+'_top_off']
                if mass_type.lower() in ['prop','propellant','mp']:
                    unknowns[target_name+'_propellant_mass_delta_'+base_name] =
                    ↪  params[self.base_name+'_dm']
                elif mass_type.lower() in ['inert','fixed','structure','mi']:
                    unknowns[target_name+'_inert_mass_delta_'+base_name] =
                    ↪  params[self.base_name+'_dm']
                for subidx in self.sub:
                    unknowns[target_name+'_inert_mass_delta_'+base_name] -=
                    ↪  params[target_name+'sub'+str(subidx)+'_inert_mass']
```

## *I.12 Fluids Definitions Model*

```
# -*- coding: utf-8 -*-
"""
Description: helper functions for fluid definitions for the propulsion and
thermal subsystem models

Written by:
    Douglas J. Trent
    NASA Marshall Space Flight Center
    Advanced Concept Office
    douglas.trent@nasa.gov

Created: 04/03/2017
Revised: 04/06/2017
"""
# import DYREQT Subelement base class
from Constants import R as R_PQ
from math import log, exp
from scipy.optimize import brentq

R = R_PQ.value

# units in the propellant defs:
#   liquid_density - kg/m**3
#   M - g/mol
#   Tc - K
#   Pc - bar
#   Hvap - kJ/kg
#   Tref - K
#   Pref - Atm.
fluid_defs = {
   'o2':{'type':'cryogenic','alias':['lox'],'liquid_density':1141.0,'M':31.9988,'Tc':154.58,
         'Pc':50.0343,'air_CF':None,'Hvap':212.9,'Tref':90.,'Pref':1.0,'Cp_coeff':None},
   'h2':{'type':'cryogenic','alias':['lh2'],'liquid_density':70.8,'M':2.016,'Tc':33.145,
         'Pc':12.964,'air_CF':None,'Hvap':461.0,'Tref':20.,'Pref':1.0,
         'Cp_coeff':{⌋
         ↪  1000:[33.066178,-11.363417,11.432816,-2.772874,-0.158558,-9.980797,172.707974,0.],
            2500:[18.563083,12.257357,-2.859786,0.268238,1.977990,-1.147438,156.288133,0.],
            6000:[43.413560,-4.293079,1.272428,-0.096876,-20.533862,-38.515158,162.081354]}},
   'ch4':{'type':'cryogenic','alias':['lch4','methane'],'liquid_density':421.0,'M':16.0425,
          'Tc':190.6,'Pc':46.1,'air_CF':None,'Hvap':537.5,'Tref':111.,'Pref':1.0,
          'Cp_coeff':{⌋
          ↪  1300:[-0.703029,108.4773,-42.52157,5.862788,0.678565,-76.84376,158.713,-74.87310],
            6000:[85.81217,11.26467,-2.114146,0.138190,-26.42221,-153.5327,224.4143,-74.87310]}},
   'mon':{'type':'storable','alias':[],'liquid_density':1370.0,'M':None,'Tc':None,
          'Pc':None,'air_CF':None,'Hvap':None,'Tref':None,'Pref':None,'Cp_coeff':None},
   'udmh':{'type':'storable','alias':[],'liquid_density':793.0,'M':60.0983,'Tc':523.15,
           'Pc':54.2,'air_CF':None,'Hvap':None,'Tref':None,'Pref':None,'Cp_coeff':None},
   'mmh':{'type':'storable','alias':[],'liquid_density':880.0,'M':46.0717,'Tc':585.15,
          'Pc':82.4,'air_CF':None,'Hvap':None,'Tref':None,'Pref':None,'Cp_coeff':None},
   'n2o':{'type':'storable','alias':['nitrousoxide'],'liquid_density':912.0,'M':44.013,'Tc':309.55,
          'Pc':72.38,'air_CF':None,'Hvap':None,'Tref':None,'Pref':None,'Cp_coeff':None},
   'n2o4':{'type':'storable','alias':['nto'],'liquid_density':1450.0,'M':92.0110,'Tc':431.,
           'Pc':101.,'air_CF':None,'Hvap':None,'Tref':None,'Pref':None,'Cp_coeff':None},
   'nitrcacid':{'type':'storable','alias':[],'liquid_density':1510.0,'M':63.0128,'Tc':None,
                'Pc':None,'air_CF':None,'Hvap':None,'Tref':None,'Pref':None,'Cp_coeff':None},
   'rp1':{'type':'storable','alias':['kerosene'],'liquid_density':810.0,'M':None,'Tc':None,
          'Pc':None,'air_CF':4.5,'Hvap':None,'Tref':None,'Pref':None,'Cp_coeff':None},
   'n2h4':{'type':'monoprop','alias':['hydrazine'],'liquid_density':1008.0,'M':32.0452,'Tc':653.15,
           'Pc':14.186,'air_CF':None,'Hvap':None,'Tref':None,'Pref':None,'Cp_coeff':None},
   'h2o2':{'type':'monoprop','alias':['hydrogenperoxide'],'liquid_density':1390.0,'M':34.0147,
           'Tc':728.0,'Pc':220.0,'air_CF':None,'Hvap':None,'Tref':None,'Pref':None,'Cp_coeff':None},
   'isopropylnitrate':{'type':'monoprop','alias':[],'liquid_density':1040.0,'M':105.0926,'Tc':None,
                       'Pc':None,'air_CF':None,'Hvap':None,'Tref':None,'Pref':None,'Cp_coeff':None},
   'he':{'type':'pressurant','alias':['helium'],'liquid_density':None,'M':4.0026,'Tc':5.195,
         'Pc':2.275,'air_CF':None,'Hvap':20.7,'Tref':4.2,'Pref':1.0,'Cp_coeff':None},
   'n2':{'type':'pressurant','alias':['nitrogen'],'liquid_density':None,'M':28.0134,'Tc':126.192,
         'Pc':3.3958,'air_CF':None,'Hvap':199.2,'Tref':77.4,'Pref':1.0,'Cp_coeff':None},
   'xe':{'type':'electric','alias':['xenon'],'liquid_density':None,'M':131.293,'Tc':289.74,
         'Pc':58.42,'air_CF':None,'Hvap':96.3,'Tref':165.,'Pref':1.0,'Cp_coeff':None},
   'ar':{'type':'electric','alias':['argon'],'liquid_density':None,'M':39.948,'Tc':150.86,
         'Pc':48.63,'air_CF':None,'Hvap':162.7,'Tref':87.3,'Pref':1.0,'Cp_coeff':None},
   'kr':{'type':'electric','alias':['krypton'],'liquid_density':None,'M':83.798,'Tc':209.48,
         'Pc':55.25,'air_CF':None,'Hvap':107.6,'Tref':391.,'Pref':1.0,'Cp_coeff':None},
}

def check_def(fluid):
    """Check for the existence of the given fluid in the fluid def

    Args
    ----
    fluid : str
        The string name of the fluid to check.

    Returns
    -------
    res : bool
        The result of the definitions check. True if the fluid exists, False
        otherwise.

    fluid_def : dict
        The fluid definition
    """

    res = False
    fluid_def = dict()

    for name,mdata in fluid_defs.items():
        if fluid.lower() in mdata['alias']+[name]:
           res = True
           fluid_def = mdata
           break

    return res, fluid_def

def tvap(fluid,pressure):
    """Returns the vaporization temperature (boiling temp) of the fluid at
    the given pressure

    Args
    ----
    fluid : str
        The string name of the fluid

    pressure : float
        The pressure to calculate the vapor temperature at (MPa)
    Returns
    -------
    temp : float
        The vaporization temperature (K)
    """

    def_flag,fluid_def = check_def(fluid)

    if def_flag:
        # extract required params
        Pref = fluid_def['Pref'] * 0.101325 # MPa
        Tref = fluid_def['Tref']
        Hvap = fluid_def['Hvap'] * 1000. # J/kg
        M = fluid_def['M'] * 1e-3 # kg/mol
        Rspec = R / M # J/kg-K
        # Clausius-Clapeyron Equation
        temp = (Rspec * log(Pref/pressure) / Hvap + (1/Tref))**-1
    else:
        raise Exception('"{0}" is not a defined fluid'.format(fluid))

    return temp

def get_property(fluid,name):
    """Get fluid properties from the fluid definition

    Args
    ----
    fluid : str
        The string name of the fluid

    name : str
        The property name to return

    Returns : variable
        The property value of interest. If the property does not exist, a None
        value will be returned.
    """

    def_flag,fluid_def = check_def(fluid)

    if def_flag:
        if name in fluid_def.keys():
            val = fluid_def[name]
        else:
            val = None
    else:
        raise Exception('"{0}" is not a defined fluid'.format(fluid))

    return val

def pvap(fluid,temp):
    """Returns the vapor pressure of the fluid at the given temperature

    Args
    ----
    fluid : str
        The string name of the fluid

    temp : float
        The temperature to calculate the vapor pressure at (K)

    Returns
    -------
    pressure : float
        The vapor pressure of the fluid (MPa)
    """
    def_flag,fluid_def = check_def(fluid)

    if def_flag:
        # extract required params
        Pref = fluid_def['Pref'] * 0.101325 # MPa
        Tref = fluid_def['Tref']
        Hvap = fluid_def['Hvap'] * 1000. # J/kg
        M = fluid_def['M'] * 1e-3 # kg/mol
        Rspec = R / M # J/kg-K
        # Clausius-Clapeyron Equation
        pressure = Pref * exp(Hvap/Rspec*(Tref**-1 - temp**-1))
    else:
        raise Exception('"{0}" is not a defined fluid'.format(fluid))

    return pressure

def rspec(fluid):
    """Returns the specific gas constant for a fluid

    Args
    ----
    fluid : str
        The string name of the fluid

    Returns
    -------
    r_spec : float
        The specific gas constant of the fluid (J/kg-K)
    """

    def_flag,fluid_def = check_def(fluid)

    if def_flag:
        # extract required params
        M = fluid_def['M'] * 1e-3 # kg/mol
        r_spec = R / M # J/kg-K
    else:
        raise Exception('"{0}" is not a defined fluid'.format(fluid))

    return r_spec

def _vanderwaals(rho,T,P,Tc,Pc,M):
    """Van Der Waals equation for estimating the density of a fluid. Does
    not perform well near the critical temperature. A fixed point iterator is
    used to solve the equation for the density.

    Args
    ----
    rho : float
        The fluid density (kg/m^3)

    T : float
        The temperature of the fluid (K)

    P : float
        The pressure of the fluid (MPa)

    Tc : float
        The critical temperature of the fluid (K)

    Pc : float
        The critical pressure of the fluid (bar)

    M : float
        The molar mass of the fluid (g/mol)
    """

    # convert units
    M = M / 1000. # g/mol -> kg/mol
    Pc = Pc*1e5 # bar -> Pa
    P = P*1e6 # MPa -> Pa

    Vm = M / rho
    a = (27*(R*Tc)**2) / (64*Pc)
    b = (R*Tc) / (8*Pc)

    return (P + a / Vm**2) * (Vm - b) - R*T

def density(fluid,liquid=True,T=293.,P=0.344738):
    """Calculates the fluid density of a given propellant in the fluid
    definitions.

    Args
    ----
    fluid : str
        The fluid density of interest

    liquid : bool(True,False)
        Return the liquid liquid density of the propellant if True, else return
        the gas density at the specified state

    T : float(293.0)
        Temperature to calculate propellant density at, in degrees Kelvin

    P : float(0.344738)
        Pressure to calculate propellant density at, in Megapascals

    Returns
    -------
    rho : float
        Calculated fluid density (kg/m^3)
    """

    def_flag,fluid_def = check_def(fluid)

    if def_flag:
        # extract required params
        Tc = fluid_def['Tc']
        Pc = fluid_def['Pc']
        M = fluid_def['M']
        air_cf = fluid_def['air_CF']
        rho_l = fluid_def['liquid_density']
        if liquid:
            if rho_l:
                rho = rho_l
            else:
                raise Exception('liquid density not defined for fluid "{0}"'.format(fluid))
        else:
            if Tc and Pc and M:
                rho = brentq(_vanderwaals, 0.1, 5000., args=(T,P,Tc,Pc,M)) # returned density is in
                ↪  kg/m**3
            elif air_cf:
                rho = []
                for prop in ['o2','n2','ar']: # air
                    Tc = fluid_defs[prop]['Tc']
                    Pc = fluid_defs[prop]['Pc']
                    M = fluid_defs[prop]['M']
                    rho.append(brentq(_vanderwaals, 0.1, 5000., args=(T,P,Tc,Pc,M))) # returned
                    ↪  density is in kg/m**3
                rho = sum([rho[i]*fraction for i,fraction in enumerate([0.21,0.78,0.01])])*air_cf
            else:
                raise Exception('insufficient definition for density calculation of fluid
                ↪  "{0}"'.format(fluid))

    else:
        raise Exception('"{0}" is not a defined fluid'.format(fluid))
    return rho

def heat_ratio(fluid,temp):
    """Calculates the fluid specific heat ratio at a temperature

    Args
    ----
    fluid : str
        The fluid density of interest

    temp : float(293.0)
        Temperature to calculate fluid specific heat ratio at (K)

    Returns
    -------
    gama : float
        The specific heat ratio
    """

    # don't need to check first, other functions will do that

    # extract required params
    Cp = cp(fluid,temp) # J/kg-K
    Rspec = rspec(fluid) # J/kg-K

    # calc
    gama = Cp / (Cp - Rspec)

    return gama


def cp(fluid,T):
    """Calculates the fluid specific heat at a temperature

    Args
    ----
    fluid : str
        The fluid density of interest

    T : float(293.0)
        Temperature to calculate fluid specific heat at (K)

    Returns
    -------
    Cp : float
        The specific heat (J/kg-K)
    """

    def_flag,fluid_def = check_def(fluid)

    # extract required params
    Cp_coeff = fluid_def['Cp_coeff']
    M = fluid_def['M']

    if not def_flag:
        msg = ('{0} is not a defined fluid'.format(fluid))
        raise Exception(msg)

    if not Cp_coeff:
        msg = ('{0} does not have defined coefficients required to calculate '
               'specific heat'.format(fluid))
        raise Exception(msg)

    # equation from NIST webbook
    T_flag = False
    for T_max,coeff in Cp_coeff.items():
        if T <= T_max:
            A = coeff[0]
            B = coeff[1]
            C = coeff[2]
            D = coeff[3]
            E = coeff[4]
            T_flag = True
            break
    if not T_flag:
        msg = ('cannot calculate specific heat for T={0} K. Tmax={1}'.format(T,max(Cp_coeff.keys())))
        raise Exception(msg)

    t = T/1000.
    Cp =  A + B*t + C*t**2 + D*t**3 + E/t**2 # J/mol-K

    Cp = Cp / M * 1000. # J/kg-K

    return Cp # J/kg-K
```

## I.13 Costing Model

```
# -*- coding: utf-8 -*-
"""
Description: functions for estimating cost of space transportation vehicles
based on the TRANSCOST v7.1 model

Written by:
    Douglas J. Trent
    NASA Marshall Space Flight Center
    Advanced Concept Office
    douglas.trent@nasa.gov

Created:05/16/2017
Revised:08/25/2017
"""
from math import log, exp

def collect_inps(sizing_outs):

    # initialize cost inputs
    cost_ins = {'num_stages':None,
                's1_class':None,
                's1_propellant':None,
                's1_prop_mass':None,
                's1_num_engines':None,
                's1_feed':None,
                's1_engines_mass':None,
                's1_engine_mass':None,
                's1_veh_manned':None,
                's1_veh_noengines_dry_mass':None,
                's1_veh_burnout_mass':None,
                's2_class':None,
                's2_propellant':None,
                's2_prop_mass':None,
                's2_num_engines':None,
                's2_feed':None,
                's2_engines_mass':None,
                's2_engine_mass':None,
                's2_veh_manned':None,
                's2_veh_noengines_dry_mass':None,
                's2_veh_burnout_mass':None,
                's3_class':None,
                's3_propellant':None,
                's3_prop_mass':None,
                's3_num_engines':None,
                's3_feed':None,
                's3_engines_mass':None,
                's3_engine_mass':None,
                's3_veh_manned':None,
                's3_veh_noengines_dry_mass':None,
                's3_veh_burnout_mass':None,
                'engine_tech':False,
                'veh_tech':False
                }

    # set cost inputs from sizing outputs
    # veh inputs
    cost_ins['num_stages'] = sizing_outs['Num Stages']
    # stage 1
    cost_ins['s1_class'] = sizing_outs['Stage 1 MPS Class']
    if sizing_outs['Stage 1 MPS Propellants'].find('/') >= 0:
        # biprops
        if any([True for prop in ['lox','lh2','lch4'] if prop in sizing_outs['Stage 1 MPS
        ↪  Propellants']]):
            # cryo
            cost_ins['s1_propellant'] = 'cryo'
        else:
            # storable
            cost_ins['s1_propellant'] = 'storable'
    else:
        # monoprop
        cost_ins['s1_propellant'] = 'monoprop'
    cost_ins['s1_prop_mass'] = sizing_outs['Stage 1 MPS Propellant Mass(kg)'] + sizing_outs['Stage 1
    ↪  RCS Propellant Mass(kg)']
    if cost_ins['s1_class'] == 'liquid':
        num_engines = 4
        if cost_ins['s1_propellant'] == 'cryo':
            feed = 'pump'
        else:
            feed = 'pressure'
    elif cost_ins['s1_class'] == 'electric':
        num_engines = 1
        feed = 'pressure'
    elif cost_ins['s1_class'] == 'nuclear':
        num_engines = 3
        feed = 'pump'
    elif cost_ins['s1_class'] == 'solid':
        num_engines = 1
        feed = None
    cost_ins['s1_num_engines'] = num_engines
    cost_ins['s1_feed'] = feed
    cost_ins['s1_engines_mass'] = sizing_outs['Stage 1 Engines Mass(kg)']
    cost_ins['s1_engine_mass'] = cost_ins['s1_engines_mass'] / num_engines
    try:
        cost_ins['s1_veh_manned'] = sizing_outs['Stage 1 Structures Type']
    except KeyError:
        cost_ins['s1_veh_manned'] = 'unmanned'
    cost_ins['s1_veh_noengines_dry_mass'] = sum([sizing_outs['Stage 1 Avionics Mass(kg)'],
                                                 sizing_outs['Stage 1 Power Mass(kg)'],
                                                 sizing_outs['Stage 1 Structures Mass(kg)'],
                                                 sizing_outs['Stage 1 Tanks Mass(kg)'],
                                                 sizing_outs['Stage 1 Thermal Mass(kg)']])
    cost_ins['s1_veh_burnout_mass'] = sizing_outs['Stage 1 Burnout Mass(kg)']
    # stage 2
    if cost_ins['num_stages'] > 1:
        cost_ins['s2_class'] = sizing_outs['Stage 2 MPS Class']
        if sizing_outs['Stage 2 MPS Propellants'].find('/') >= 0:
            # biprops
            if any([True for prop in ['lox','lh2','lch4'] if prop in sizing_outs['Stage 2 MPS
            ↪  Propellants']]):
                # cryo
                cost_ins['s2_propellant'] = 'cryo'
            else:
                # storable
                cost_ins['s2_propellant'] = 'storable'
        else:
            # monoprop
            cost_ins['s2_propellant'] = 'monoprop'
        cost_ins['s2_prop_mass'] = sizing_outs['Stage 2 MPS Propellant Mass(kg)'] + sizing_outs['Stage
        ↪  2 RCS Propellant Mass(kg)']
        if cost_ins['s2_class'] == 'liquid':
            num_engines = 4
            if cost_ins['s2_propellant'] == 'cryo':
                feed = 'pump'
            else:
                feed = 'pressure'
        elif cost_ins['s2_class'] == 'electric':
            num_engines = 1
            feed = 'pressure'
        elif cost_ins['s2_class'] in ['nuclear','massless']:
            num_engines = 3
            feed = 'pump'
        elif cost_ins['s2_class'] == 'solid':
            num_engines = 1
            feed = None
        cost_ins['s2_num_engines'] = num_engines
        cost_ins['s2_feed'] = feed
        cost_ins['s2_engines_mass'] = sizing_outs['Stage 2 Engines Mass(kg)']
        cost_ins['s2_engine_mass'] = cost_ins['s2_engines_mass'] / num_engines
        try:
            cost_ins['s2_veh_manned'] = sizing_outs['Stage 2 Structures Type']
        except KeyError:
            cost_ins['s2_veh_manned'] = 'unmanned'
        if cost_ins['s2_class'] == 'massless':
            cost_ins['s2_veh_noengines_dry_mass'] = sum([sizing_outs['Stage 2 Power Mass(kg)'],
                                                         sizing_outs['Stage 2 Structures Mass(kg)'],
                                                         sizing_outs['Stage 2 Tanks Mass(kg)'],
                                                         sizing_outs['Stage 2 Thermal Mass(kg)']])
        else:
            cost_ins['s2_veh_noengines_dry_mass'] = sum([sizing_outs['Stage 2 Avionics Mass(kg)'],
                                                         sizing_outs['Stage 2 Power Mass(kg)'],
                                                         sizing_outs['Stage 2 Structures Mass(kg)'],
                                                         sizing_outs['Stage 2 Tanks Mass(kg)'],
                                                         sizing_outs['Stage 2 Thermal Mass(kg)']])
        cost_ins['s2_veh_burnout_mass'] = sizing_outs['Stage 2 Burnout Mass(kg)']
    # stage 3
    if cost_ins['num_stages'] > 2:
        cost_ins['s3_class'] = sizing_outs['Stage 3 MPS Class']
        if sizing_outs['Stage 3 MPS Propellants'].find('/') >= 0:
            # biprops
            if any([True for prop in ['lox','lh2','lch4'] if prop in sizing_outs['Stage 3 MPS
            ↪  Propellants']]):
                # cryo
                cost_ins['s3_propellant'] = 'cryo'
            else:
                # storable
                cost_ins['s3_propellant'] = 'storable'
        else:
            # monoprop
            cost_ins['s3_propellant'] = 'monoprop'
        cost_ins['s3_prop_mass'] = sizing_outs['Stage 3 MPS Propellant Mass(kg)'] + sizing_outs['Stage
        ↪  3 RCS Propellant Mass(kg)']
        if cost_ins['s3_class'] == 'liquid':
            num_engines = 4
            if cost_ins['s3_propellant'] == 'cryo':
                feed = 'pump'
            else:
                feed = 'pressure'
        elif cost_ins['s3_class'] == 'electric':
            num_engines = 1
            feed = 'pressure'
        elif cost_ins['s3_class'] in ['nuclear','massless']:
            num_engines = 3
            feed = 'pump'
        elif cost_ins['s3_class'] == 'solid':
            num_engines = 1
            feed = None
        cost_ins['s3_num_engines'] = num_engines
        cost_ins['s3_feed'] = feed
        cost_ins['s3_engines_mass'] = sizing_outs['Stage 3 Engines Mass(kg)']
        cost_ins['s3_engine_mass'] = cost_ins['s3_engines_mass'] / num_engines
        try:
            cost_ins['s3_veh_manned'] = sizing_outs['Stage 3 Structures Type']
        except KeyError:
            cost_ins['s3_veh_manned'] = 'unmanned'
        if cost_ins['s3_class'] == 'massless':
            cost_ins['s3_veh_noengines_dry_mass'] = sum([sizing_outs['Stage 3 Power Mass(kg)'],
                                                         sizing_outs['Stage 3 Structures Mass(kg)'],
                                                         sizing_outs['Stage 3 Tanks Mass(kg)'],
                                                         sizing_outs['Stage 3 Thermal Mass(kg)']])
        else:
            cost_ins['s3_veh_noengines_dry_mass'] = sum([sizing_outs['Stage 3 Avionics Mass(kg)'],
                                                         sizing_outs['Stage 3 Power Mass(kg)'],
                                                         sizing_outs['Stage 3 Structures Mass(kg)'],
                                                         sizing_outs['Stage 3 Tanks Mass(kg)'],
                                                         sizing_outs['Stage 3 Thermal Mass(kg)']])
        cost_ins['s3_veh_burnout_mass'] = sizing_outs['Stage 3 Burnout Mass(kg)']
    # techs
    othertechs = [sizing_outs['Wireless Sensors'],
                  sizing_outs['Composite Structures'],
                  sizing_outs['Composite Tanks'],
                  sizing_outs['Integrated MPS/RCS Prop'],
                  sizing_outs['Active Cryo Cooling']
                 ]
    if sizing_outs['Low Leak Valves']:
        cost_ins['engine_tech'] = False
    else:
        cost_ins['engine_tech'] = True
    if sizing_outs['High Capacity Energy Storage'] == True:
        cost_ins['veh_tech'] = True
    elif any([tech for tech in othertechs]):
        cost_ins['veh_tech'] = True

    return cost_ins

def estimate_costs(cost_ins):

    # initialize outputs
    cost_outs = {'Vehicle Gross Cost(MYr)':0.,
                 'Vehicle Gross Prod Cost(MYr)':0.,
                 'Vehicle Gross Dev Cost(MYr)':0.,
                 'Stage 1 Gross Cost(MYr)':0.,
                 'Stage 1 Gross Prod Cost(MYr)':0.,
                 'Stage 1 Engines Prod Cost(MYr)':0.,
                 'Stage 1 Vehicle Prod Cost(MYr)':0.,
                 'Stage 1 Gross Dev Cost(MYr)':0.,
                 'Stage 1 Engines Dev Cost(MYr)':0.,
                 'Stage 1 Vehicle Dev Cost(MYr)':0.,
                 'Stage 2 Gross Cost(MYr)':0.,
                 'Stage 2 Gross Prod Cost(MYr)':0.,
                 'Stage 2 Engines Prod Cost(MYr)':0.,
                 'Stage 2 Vehicle Prod Cost(MYr)':0.,
                 'Stage 2 Gross Dev Cost(MYr)':0.,
                 'Stage 2 Engines Dev Cost(MYr)':0.,
                 'Stage 2 Vehicle Dev Cost(MYr)':0.,
                 'Stage 3 Gross Cost(MYr)':0.,
                 'Stage 3 Gross Prod Cost(MYr)':0.,
                 'Stage 3 Engines Prod Cost(MYr)':0.,
                 'Stage 3 Vehicle Prod Cost(MYr)':0.,
                 'Stage 3 Gross Dev Cost(MYr)':0.,
                 'Stage 3 Engines Dev Cost(MYr)':0.,
                 'Stage 3 Vehicle Dev Cost(MYr)':0.,
                 }

    # cost stage 1
    # engines
    (cost_outs['Stage 1 Engines Dev Cost(MYr)'],
     cost_outs['Stage 1 Engines Prod Cost(MYr)']) = cost_engines(cost_ins['s1_engine_mass'],
                                                                 cost_ins['engine_tech'],
                                                                 cost_ins['s1_num_engines'],
                                                                 cost_ins['s1_class'],
                                                                 cost_ins['s1_feed'],
                                                                 cost_ins['s1_propellant'])
    # vehicle
    (cost_outs['Stage 1 Vehicle Dev Cost(MYr)'],
     cost_outs['Stage 1 Vehicle Prod Cost(MYr)']) = cost_stage(cost_ins['s1_veh_noengines_dry_mass'],
                                                               cost_ins['s1_veh_burnout_mass'],
                                                               cost_ins['s1_engines_mass'],
                                                               cost_ins['s1_prop_mass'],
                                                               cost_ins['veh_tech'],
                                                               cost_ins['s1_class'],
                                                               cost_ins['s1_veh_manned'],
                                                               cost_ins['s1_propellant'])
    # totals
    cost_outs['Stage 1 Gross Dev Cost(MYr)'] = (cost_outs['Stage 1 Engines Dev Cost(MYr)'] +
                                                cost_outs['Stage 1 Vehicle Dev Cost(MYr)'])
    cost_outs['Stage 1 Gross Prod Cost(MYr)'] = (cost_outs['Stage 1 Engines Prod Cost(MYr)'] +
                                                 cost_outs['Stage 1 Vehicle Prod Cost(MYr)'])
    cost_outs['Stage 1 Gross Cost(MYr)'] = (cost_outs['Stage 1 Gross Dev Cost(MYr)'] +
                                            cost_outs['Stage 1 Gross Prod Cost(MYr)'])

    if cost_ins['num_stages'] > 1:
        # cost stage 2
        # engines
        (cost_outs['Stage 2 Engines Dev Cost(MYr)'],
         cost_outs['Stage 2 Engines Prod Cost(MYr)']) = cost_engines(cost_ins['s2_engine_mass'],
                                                                     cost_ins['engine_tech'],
                                                                     cost_ins['s2_num_engines'],
                                                                     cost_ins['s2_class'],
                                                                     cost_ins['s2_feed'],
                                                                     cost_ins['s2_propellant'])
        # vehicle
        (cost_outs['Stage 2 Vehicle Dev Cost(MYr)'],
         cost_outs['Stage 2 Vehicle Prod Cost(MYr)']) =
         ↪  cost_stage(cost_ins['s2_veh_noengines_dry_mass'],
                                                                   cost_ins['s2_veh_burnout_mass'],
                                                                   cost_ins['s2_engines_mass'],
                                                                   cost_ins['s2_prop_mass'],
                                                                   cost_ins['veh_tech'],
                                                                   cost_ins['s2_class'],
                                                                   cost_ins['s2_veh_manned'],
                                                                   cost_ins['s2_propellant'])
        #totals
        cost_outs['Stage 2 Gross Dev Cost(MYr)'] = (cost_outs['Stage 2 Engines Dev Cost(MYr)'] +
                                                    cost_outs['Stage 2 Vehicle Dev Cost(MYr)'])
        cost_outs['Stage 2 Gross Prod Cost(MYr)'] = (cost_outs['Stage 2 Engines Prod Cost(MYr)'] +
                                                     cost_outs['Stage 2 Vehicle Prod Cost(MYr)'])
        cost_outs['Stage 2 Gross Cost(MYr)'] = (cost_outs['Stage 2 Gross Dev Cost(MYr)'] +
                                                cost_outs['Stage 2 Gross Prod Cost(MYr)'])

    if cost_ins['num_stages'] > 2:
        # cost stage 3
        # engines
        (cost_outs['Stage 3 Engines Dev Cost(MYr)'],
         cost_outs['Stage 3 Engines Prod Cost(MYr)']) = cost_engines(cost_ins['s3_engine_mass'],
                                                                     cost_ins['engine_tech'],
                                                                     cost_ins['s3_num_engines'],
                                                                     cost_ins['s3_class'],
                                                                     cost_ins['s3_feed'],
                                                                     cost_ins['s3_propellant'])
        # vehicle
        (cost_outs['Stage 3 Vehicle Dev Cost(MYr)'],
         cost_outs['Stage 3 Vehicle Prod Cost(MYr)']) =
         ↪  cost_stage(cost_ins['s3_veh_noengines_dry_mass'],
                                                                   cost_ins['s3_veh_burnout_mass'],
                                                                   cost_ins['s3_engines_mass'],
                                                                   cost_ins['s3_prop_mass'],
                                                                   cost_ins['veh_tech'],
                                                                   cost_ins['s3_class'],
                                                                   cost_ins['s3_veh_manned'],
                                                                   cost_ins['s3_propellant'])
        #totals
        cost_outs['Stage 3 Gross Dev Cost(MYr)'] = (cost_outs['Stage 3 Engines Dev Cost(MYr)'] +
                                                    cost_outs['Stage 3 Vehicle Dev Cost(MYr)'])
        cost_outs['Stage 3 Gross Prod Cost(MYr)'] = (cost_outs['Stage 3 Engines Prod Cost(MYr)'] +
                                                     cost_outs['Stage 3 Vehicle Prod Cost(MYr)'])
        cost_outs['Stage 3 Gross Cost(MYr)'] = (cost_outs['Stage 3 Gross Dev Cost(MYr)'] +
                                                cost_outs['Stage 3 Gross Prod Cost(MYr)'])

    # cost totals
    (cost_outs['Vehicle Gross Dev Cost(MYr)'],
     cost_outs['Vehicle Gross Prod Cost(MYr)'],
     cost_outs['Vehicle Gross Cost(MYr)']) = cost_veh(cost_ins['num_stages'],
                                                      cost_outs['Stage 1 Gross Dev Cost(MYr)'],
                                                      cost_outs['Stage 1 Gross Prod Cost(MYr)'],
                                                      cost_outs['Stage 2 Gross Dev Cost(MYr)'],
                                                      cost_outs['Stage 2 Gross Prod Cost(MYr)'],
                                                      cost_outs['Stage 3 Gross Dev Cost(MYr)'],
                                                      cost_outs['Stage 3 Gross Prod Cost(MYr)'])

    return cost_outs

def cost_stage(mass_dry,mass_burnout,mass_engines,mass_prop,tech,stage_class,manned,prop):

    dev = 0.
    prod = 0.
    M  = mass_dry # in kg
    Mn = mass_burnout # in kg
    Me = mass_engines # in kg
    Mp = mass_prop # in kg

    # set factors
    n = 1  # number of items produced
    if tech:
        f1 = 1.2 # development standards factor, assumes some new technical/operational features
    else:
        f1 = 1.0 # development standards factor, state of the art design similar to current designs
    f3 = 0.8 # team experience factor, team has performed development of similar project
    p = 0.0113 * log(M) + 0.852 # learning factor, production rate of 2 stages per year
    if p > 1:
        p = 1
    f4 = (1/n) * sum([i**(log(p)/log(2)) for i in range(1,n+1)]) # production cost reduction factor

    if manned:
        # crewed space system
        f2 = 1. # technical quality factor
        x_dev = 0.37 #  development CER sensitivity factor for
        a_dev = 1220. #  development CER scaling factor
        x_prod = 0.98 #  production CER sensitivity factor for
        a_prod = 0.16 #  production CER scaling factor
    else:
        if stage_class == 'solid':
            # prop module
            f2 = 1. # technical quality factor
            x_dev = 0.55 #  development CER sensitivity factor for
            a_dev = 15.4 #  development CER scaling factor
            x_prod = 0.49 #  production CER sensitivity factor for
            a_prod = 4.65 #  production CER scaling factor
        else:
            # expendable ballistic stage
            x_dev = 0.555 #  development CER sensitivity factor for
            a_dev = 98.6 #  development CER scaling factor
            x_prod = 0.65 #  production CER sensitivity factor for
            keff = (Mn - Me) / Mp # specific net mass fraction
            if prop == 'cryo':
                # average reference net mass fraction for cryo stages
                kref = exp(-1.471159 - 0.2061181*log(Mp/1e3) +
                           0.0459295*(log(Mp/1e3)-3.98104)**2 -
                           0.0053711*(log(Mp/1e3)-3.98104)**3)
                f2 = kref / keff # technical quality factor
                a_prod = 1.30 #  production CER scaling factor
            else:
                # average reference net mass fraction for storable stages
                kref = exp(-1.257492 - 0.3925932*log(Mp/1e3) +
                           0.0486939*(log(Mp/1e3)-2.86218)**2 -
                           0.0006765*(log(Mp/1e3)-2.86218)**3)
                f2 = kref / keff # technical quality factor
                a_prod = 0.83 #  production CER scaling factor

    # cost stage development
    dev = a_dev * M ** x_dev * f1 * f2 * f3

    # cost stage production
    prod = a_prod * n * M ** x_prod * f4

    return dev,prod

def cost_engines(mass,tech,num_engines,stage_class,feed,prop):

    dev = 0.
    prod = 0.
    M = mass # in kg

    # set factors
    n = num_engines  # number of items produced
    if tech:
        f1 = 1.2 # development standards factor, assumes some new technical/operational features
    else:
        f1 = 1.0 # development standards factor, state of the art design similar to current designs
    f3 = 0.8 # team experience factor, team has performed development of similar project
    p = 0.0126 * log(M) + 0.8037 # learning factor, production rate of 10 engines per year
    if p > 1:
        p = 1
    f4 = (1/n) * sum([i**(log(p)/log(2)) for i in range(1,n+1)]) # production cost reduction factor

    if stage_class == 'solid':
        f2 = 1. # technical quality factor
        x_dev = 0.53 #  development CER sensitivity factor for
        a_dev = 19.2 #  development CER scaling factor
        x_prod = 0.395 #  production CER sensitivity factor for
        a_prod = 2.42 #  production CER scaling factor
    else:
        if feed == 'pump':
            x_dev = 0.52 #  development CER sensitivity factor for
            a_dev = 1975.0 #  development CER scaling factor
            if prop == 'cryo':
                x_prod = 0.45 #  production CER sensitivity factor for
                a_prod = 5.16 #  production CER scaling factor
            elif prop == 'storable':
                x_prod = 0.535 #  production CER sensitivity factor for
                a_prod = 1.9 #  production CER scaling factor
            elif prop == 'monoprop':
                # nuclear engine, use cryo settings
                x_prod = 0.45 #  production CER sensitivity factor for
                a_prod = 5.16 #  production CER scaling factor
            if stage_class == 'nuclear':
                Nq = 10 # number of dev and qual
            else:
                # all others (liquids, electric)
                Nq = 500 # number of dev and qual
        else:
            x_dev = 0.365 #  development CER sensitivity factor for
            a_dev = 155.0 #  development CER scaling factor
            if prop == 'storable':
                x_prod = 0.535 #  production CER sensitivity factor for
                a_prod = 1.9 #  production CER scaling factor
            elif prop == 'monoprop':
                x_prod = 0.535 #  production CER sensitivity factor for
                a_prod = 1.13 #  production CER scaling factor
            Nq = 500 # number of dev and qual
        f2 = 0.026 * log(Nq)**2 # technical quality factor

    # cost stage development
    dev = a_dev * M ** x_dev * f1 * f2 * f3

    # cost stage production
    prod = a_prod * n * M ** x_prod * f4

    return dev,prod

def cost_veh(num_stages, s1_dev_cost, s1_prod_cost, s2_dev_cost, s2_prod_cost, s3_dev_cost,
↪   s3_prod_cost):

    dev = 0.
    prod = 0.
    total = 0.

    # set factors
    f0_dev = 1.04**num_stages
    f0_prod = 1.02 # assumed minimum from handbook
    f6 = 1. # schedule cost growth factor at 100% baseline schedule
    f7 = 1. # cost growth factor for single contractor program
    f8 = 1. # productivity correction factor for USA

    # calculate dev cost
    dev = f0_dev * (s1_dev_cost + s2_dev_cost + s3_dev_cost) * f6 * f7 * f8 # MYr

    # calculate prod cost
    prod = f0_prod * num_stages * (s1_prod_cost + s2_prod_cost + s3_prod_cost) # MYr

    # calculate total cost
    total = dev + prod # MYr

    return dev,prod,total

def transcost(sizing_outs):

    # get cost inputs from sizing outputs
    cost_ins = collect_inps(sizing_outs)

    # estimate cost outputs
    cost_outs = estimate_costs(cost_ins)

    # clean up outputs
    for key in cost_outs.keys():
        if cost_outs[key] == 0:
            cost_outs[key] = None

    all_data = dict(**sizing_outs,**cost_outs)

    return all_data
```

# REFERENCES

[1] 111th Congress of the United States of America, “NASA Authorization Act of 2010,” October 2010.

[2] 114th Congress of the United States of America, “Commerce, justice, science, and related agencies appropriations bill, 2017,” March 2016.

[3] 115th Congress of the United States of America, “NASA Autherization Act of 2017,” March 2017.

[4] Alford, B. and Prince, A., “NASA project cost estimating capability: New analyses for spacecraft estimating,” 2016.

[5] Allen, W. H., “Dictionary of technical terms for aerospace use.” NASA Headquarters, 1965. NASA SP-7.

[6] Architecture, “Merriam-webster online dictionary,” June 2016. Retrieved June 21, 2016, from http://www.merriam-webster.com/dictionary/architecture.

[7] Assistant Secretary of Defense for Research and Engineering, *Technology Readiness Assessment (TRA) Guidance.* U.S. Department of Defense, April 2011. Revised 13 May 2011.

[8] Augustine, N. R., Austin, W. M., Bajmuk, B. I., Chiao, L., Chyba, C., Crawley, E. F., Greason, J. K., Kennel, C. F., Lyles, L. L., and others, *Seeking a human spaceflight program worthy of a great nation.* Review of U.S. Human Spaceflight Plans Committee, 2009.

[9] Bergenthal, J., “Final report mode-based engineering (MBE) subcommittee,” 2011. NDIA Systems Engineering Division, M&S Committee.

[10] Biltgen, P. T., Ender, T., and Mavris, D. N., “Development of a collaborative capability-based tradeoff environment for complex system architectures,” *American Institute of Aeronautics and Astronautics*, 2006.

[11] Boriah, S., Chandola, V., and Kumar, V., “Similarity measures for categorical data: A comparative evaluation,” in *Proceedings of the 2008 SIAM International Conference on Data Mining*, pp. 243–254, SIAM, 2008.

[12] Brown, C. D., *Elements of Spacecraft Design.* AIAA Education Series, American Institute of Aeronautics and Astronautics, 2002.

[13] Bush, G. W., “President Bush delivers remarks on U.S. space policy,” January 2004. Public Speech Transcripts.

[14] Campaign, "Merriam-webster online dictionary," June 2016. Retrieved June 21, 2016, from http://www.merriam-webster.com/dictionary/campaign.

[15] Chairman of the Joint Chiefs of Staff, *Manual for The Operation of the Joint Capabilities Integration and Development Systems*, February 2015. Instruction CJCSI 3170.01 I.

[16] Chapman, J. M., "Intro to rocket science: Sizing an in-space pressure regulated hypergolic propulsion system," 2016. NASA Marshall Space Flight Center. Lecture.

[17] Charania, A., "Prioritization of advanced space transportation technologies utilizing the abbreviated technology identification, evaluation, and selection (ATIES) methodology for a reusable launch vehicle (RLV)," Master's thesis, Georgia Institute of Technology, 2000.

[18] Clark, I. G. and Olds, J. R., "The technology roadmapping and investment planning system (trips)," *AIAA*, 2005.

[19] Debreceni, M. J., Lay, W. D., Jaekle Jr, D. E., and Graffer, A. C., "Design and development of the axaf-ips pmd & pmd integration," *American Institute of Aeronautics and Astronautics*, 1997.

[20] Dickerson, C. and Mavris, D., "Relational oriented systems engineering (ROSE): Preliminary report.," *2011 6th International Conference on System of Systems Engineering (SoSE)*, p. 149, 2011.

[21] Dickerson, C. and Mavris, D. N., *Architecture and Principles of Systems Engineering.* CRC Press, 2010.

[22] Dieter, G. E., *Engineering Design: A Materials and Processing Approach.* McGraw-Hill series in mechanical engineering, McGraw-Hill, 2000.

[23] Domercant, J. C., *ARC-VM: An architecture real options complexity-based valuation methodology for military systems-of-systems acquisitions.* PhD thesis, Georgia Institute of Technology, 2011.

[24] Emblemsvag, J., *Life-Cycle Costing: Using Activity-Based Costing and Monte Carlo Methods to Manage Future Costs and Risks.* John Wiley & Sons, 2003.

[25] Engler, W., Biltgen, P. T., and Mavris, D. N., "Concept selection using an interactive reconfigurable matrix of alternatives (IRMA)," in *45th AIAA Aerospace Sciences Meeting and Exhibit*, vol. 10, 2007.

[26] Exploration Systems Mission Directorate, "Lunar architecture focused trade study final report, rev a." NASA Headquarters, 2005. ESMD-RQ-0005.

[27] Fabrycky, W. J. and Blanchard, B. S., *Life-cycle Cost and Economic Analysis*. Prentice Hall, 1991.

[28] Feiler, P. H., Gluch, D. P., and Hudak, J. J., “The architecture analysis & design language (AADL): An introduction,” tech. rep., Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst, 2006.

[29] Feiler, P. H., Lewis, B., Vestal, S., and Colbert, E., *An Overview of the SAE Architecture Analysis & Design Language (AADL) Standard: A Basis for Model-Based Architecture-Driven Embedded Systems Engineering*, pp. 3–15. Boston, MA: Springer US, 2005.

[30] Feller, J. R., “Preliminary study of lunar lander descent stage active thermal control systems.” NASA Ames Research Center, 2011.

[31] Friedenthal, S., Moore, A., and Steiner, R., “OMG systems modeling language (OMG SysML) tutorial,” in *INCOSE International Symposium*, vol. 18, pp. 1731–1862, Wiley Online Library, 2008.

[32] Funaro, G. V. and Alexander, R. A., “Technology alignment and portfolio prioritization (TAPP),” *American Institute of Aeronautics and Astronautics*, 2015.

[33] Galorath, “SEER University,” 2017. http://seer-university.galorath.com/.

[34] Gass, S. I., *Decision Making, Models and Algorithms.* Krieger Publishing Comapny, 1991.

[35] Gass, S. I. and Harris, C. M., *Encyclopedia of operations research and management science.* Springer Science & Business Media, 2012.

[36] Gatian, K. N., *A Quantitative, Model-Driven Approach to Technology Selection and Development Through Epistemic Uncertainty Reduction.* PhD thesis, Georgia Institute of Technology, 2015.

[37] Gray, J., Moore, K. T., and Naylor, B. A., “OpenMDAO: An open source framework for multidisciplinary analysis and optimization,” 2010–. [Online; accessed 2016-10-18].

[38] Griendling, K. and Mavris, D., “An architecture-based approach to identifying system-of-systems alternatives,” in *System of Systems Engineering (SoSE), 2010 5th International Conference on*, IEEE, 2010.

[39] Griendling, K. A., *ARCHITECT: The Architecture-Based Technology Evaluation and Capability Tradeoff Method.* PhD thesis, Georgia Institute of Technology, 2011.

[40] Haberbusch, M., Lawless, B., Ickes, J., and Walls, L., “Reduced gravity cryo-tracker system,” in *47th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition*, p. 1599, 2009.

[41] Hall, A. D., *A Methodology for Systems Engineering.* van Nostrand, 1962.

[42] Hart, L. E., "Introduction to model-based systems engineering (MBSE) and SysML," 2015. Presented at the Delaware Valley INCOSE Chapter Meeting July 30, 2015.

[43] Hause, M. and others, "The SysML modelling language," in *Fifteenth European Systems Engineering Conference*, vol. 9, 2006.

[44] Havskjold, G., "Developing innovative products on budget and on schedule–part 1: Identifying and measuring cost drivers correlates technical uncertainty with rework cycles," in *45th AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit*, p. 5436, 2009.

[45] Havskjold, G., "Developing innovative products on budget and on schedule–part 2: Using prodecol charts to control development of an innovative advanced technology system," in *45th AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit*, p. 5437, 2009.

[46] Havskjold, G., "Developing innovative products on budget and on schedule–part 3: Generating the prodecol diagram," in *45th AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit*, p. 5437, 2009.

[47] Heineman, W., "Design mass properties ii: Mass estimating and forecasting for aerospace vehicles based on historical data," *NASA JSC-26098, Nov*, 1994.

[48] Holt, J., *UML for Systems Engineering: watching the wheels*, vol. 4. IET, 2004.

[49] Holt, J. and Perry, S., *SysML for Systems Engineering - A Model-Based Approach (2nd Edition).* Institution of Engineering and Technology, 2014.

[50] Iacobucci, J. V., *Rapid Architecture Alternative Modeling (RAAM): A Framework for Capability-Based Analysis of System of Systems Architectures.* PhD thesis, Georgia Institute of Technology, 2012.

[51] IEEE Standards Board, "Systems and software engineering – vocabulary," *ISO/IEC/IEEE 24765:2010(E)*, pp. 20,357,360, Dec 2010.

[52] International Council on Systems Engineering and Haskins, Cecilia, *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities.* International Council of Systems Engineering, 3 ed., 2006.

[53] Isakowitz, S. J., Hopkins, J. B., and Jr., J. P. H., *International Reference Guide to Space Launch Systems.* American Institute of Aeronautics and Astronautics, 4th edition ed., 2004.

[54] Joint Chiefs of Staff, *Joint Publication 1-02: Department of Defense Dictionary of Military and Associated Terms.* U.S. Department of Defense, November 2010. As amended through 15 February 2016.

[55] Joint Chiefs of Staff, *Joint Publication 3-0: Joint Operations.* U.S. Department of Defense, August 2011.

[56] Jones, E., Oliphant, T., Peterson, P., and others, "SciPy: Open source scientific tools for Python," 2001–. [Online; accessed 2016-10-18].

[57] Kirby, M. R., *A Methodology for Technology Identification, Evaluation, and Selection in Conceptual and Preliminary Aircraft Design.* PhD thesis, Georgia Institute of Technology, 2001.

[58] Koelle, D. E., *Handbook of Cost Engineering For Space Transportation Systems with Transcost 7.1 Statistical-Analytical Model for Cost Estimation and Economical Optimization of Launch Vehicles.* TransCostSystems, 2003. Report No. TCS-TR-175.

[59] Koelle, D. E., "The TransCost model for launch vehicle cost estimation and its application to future systems analysis," *Acta Astronautica*, vol. 11, no. 12, pp. 803 – 817, 1984.

[60] Komar, D., Hoffman, J., Olds, A., and Seal, M., "Framework for the parametric system modeling of space exploration architectures," in *AIAA SPACE 2008 Conference & Exposition*, p. 7845, 2008.

[61] Kravitz, S., "Packing cylinders into cylindrical containers," *Mathematics Magazine*, vol. 40, no. 2, pp. 65–71, 1967.

[62] Kujawski, E., "Analysis and critique of the system readiness level," *Systems, Man, and Cybernetics: Systems, IEEE Transactions on*, vol. 43, pp. 979–987, July 2013.

[63] Larson, W. and Pranke, L., *Human Spaceflight: Mission Analysis and Design.* McGraw-Hill Companies, October 1999.

[64] Malone, P., Smoker, R., Apgar, H., and Wolfarth, L., "The application of TRL metrics to existing cost prediction models," pp. 1–12, IEEE Publishing, March 2011.

[65] Mavris, D., "Design of experiments for practical applications in modeling, simulation, and analysis," 2011. Georgia Institute of Technology, Daniel Guggenheim School of Aerospace Engineering. Lecture.

[66] Mavris, D., "A 'paradigm shift' in complex system design: Enabling technologies for strategic decision making of advanced design concepts," 2011. Georgia Institute of Technology, Daniel Guggenheim School of Aerospace Engineering. Lecture.

[67] Mavris, D. and Griendling, K., "SoS modeling and simulation fundamentals," 2015. Georgia Institute of Technology, Daniel Guggenheim School of Aerospace Engineering. Lecture.

[68] Mavris, D. N., DeLaurentis, D. A., Bandte, O., and Hale, M. A., "A stochastic approach to multi-disciplinary aircraft analysis and design," in *36th Aerospace Sciences Meeting & Exhibit, Reno, NV*, 1998.

[69] Mavris, D. N., Dickerson, C. E., and Griendling, K., "Relational-oriented systems engineering and technology tradeoff analysis framework," *Journal of Aircraft*, vol. 50, no. 5, pp. 1564 – 1575, 2013.

[70] Mavris, D. and Griendling, K., "Relational oriented systems engineering and technology tradeoff analysis (ROSETTA) environment.," in *6th International Conference on System of Systems Engineering*, 2011.

[71] McCarter, J. and Mulqueen, J., "Computerized orbital performance analysis (COPA) user's manual," 2012.

[72] McManus, H. L., Hastings, D. E., and Warmkessel, J. M., "New methods for rapid architecture selection and conceptual design," *Journal of Spacecraft and Rockets*, vol. 41, no. 1, pp. 10–19, 2004.

[73] Michael D. Griffin, J. R. F., *Space Vehicle Design*. American Institute of Aeronautics and Astronautics, second ed., 2004.

[74] Milner, T. R., *A Risk-Informed Manufacturing Influenced Design Framework For Affordable Launch Vehicles*. PhD thesis, Georgia Institute of Technology, 2016.

[75] Mission, "Merriam-webster online dictionary," June 2016. Retrieved June 21, 2016, from http://www.merriam-webster.com/dictionary/mission.

[76] NASA, "NASA fiscal year 2005 budget request." NASA Headquarters, 2004. Retrieved June 8, 2016, from http://www.nasa.gov/about/budget/FY05_budget.html.

[77] NASA, "NASA system engineering handbook." Washington D.C., December 2007. NASA/SP-2007-6105 Rev1.

[78] NASA, "Human exploration of Mars: Design reference architecture 5.0." NASA Headquarters, 2009. SP-2009-566.

[79] NASA, "NASA fiscal year budget requests." NASA Headquarters, 2011-2018. Retrieved July 5, 2017, from http://www.nasa.gov/news/budget/index.html.

[80] NASA, "Commercial orbital transportation service: A new era in spaceflight." NASA Lyndon B. Johnson Space Center, 2014. SP-2014-617.

[81] NASA, "Human exploration of Mars: Design reference architecture 5.0, addendum 2." NASA Johnson Space Center, 2014. NASA/SP-2009-566-ADD2.

[82] NASA, "NASA space flight program and project management handbook." NASA Headquarters, 2014. NASA/SP-2014-3705.

[83] NASA, "NASA strategic plan." NASA Headquarters, 2014. NP-2014-01-964-HQ.

[84] NASA, "NASA's commercial crew program facts sheet." NASA John F. Kennedy Space Center, 2014. FS-2014-010-284-KSC.

[85] NASA, "NASA technology roadmaps: Introduction, crosscutting technologies, and index." NASA Office of the Chief Technologist, July 2015. Retrieved June 22, 2016, from http://www.nasa.gov/offices/oct/home/roadmaps/index.html.

[86] Object Managment Group, "OMG Systems Modeling Language$^{\mathrm{TM}}$(SysML®)," 2017. Accessed July 8, 2017, from http://www.omg.org/spec/SysML/index.htm.

[87] Olds, J. R., "A review of technology assessment methods for space transportation systems," in *Georgia Tech Space Systems Engineering Conference*, 2005.

[88] Olechowski, A., Eppinger, S. D., and Joglekar, N., "Technology readiness levels at 40: a study of state-of-the-art use, challenges, and opportunities," in *Management of Engineering and Technology (PICMET), 2015 Portland International Conference on*, pp. 2084–2094, IEEE, 2015.

[89] Percy, T., McGuire, M., and Polsgrove, T., "In-space transportation for NASAs evolvable Mars campaign," *AIAA SPACE Conference*, 2015.

[90] Percy, T. K., Polsgrove, T., Turpin, J., and Alexander, L., "Design and development of a methane cryogenic propulsion stage for human Mars exploration," *AIAA SPACE Conference*, 2016.

[91] Percy, T., "Human exploration architecture model (HExAM) user guide," 2015.

[92] P.J. Linstrom, W. M., ed., *NIST Chemistry WebBook, NIST Standard Reference Database Number 69.* National Institute of Standards and Technology, 2015. (Retrieved March 2017).

[93] Pugh, S., *Total design: integrated methods for successful product engineering.* Addison-Wesley Wokingham, 1991.

[94] Raymer, D. P., *Aircraft Design: A Conceptual Approach.* AIAA Education Series, American Institute of Aeronautics and Astronautics, 4th ed., 2006.

[95] Rhatigan, J. L., "Constellation program lessons learned," 2011.

[96] Ross, A. M., Diller, N., and Hastings, D., "Multi-attribute tradespace exploration with concurrent design for space system conceptual design," in *st Aerospace Sciences Meeting, AIAA2003-1328. Reno, NV: January*, pp. 6–9, 2003.

[97] Ross, A. M., Hastings, D. E., Warmkessel, J. M., and Diller, N. P., "Multi-attribute tradespace exploration as front end for effective space system design," *Journal of Spacecraft and Rockets*, vol. 41, no. 1, pp. 20–28, 2004.

[98] Ross, A. M., *Multi-attribute tradespace exploration with concurrent design as a value-centric framework for space system architecture and design*. PhD thesis, Massachusetts Institute of Technology, 2003.

[99] Sauser, B., Verma, D., Ramirez-Marquez, J., and Gove, R., "From TRL to SRL: The concept of systems readiness levels," 2006.

[100] Schankman, M. and Reynolds, J., "Advancing the art of technology cost estimating: A collaboration between NASA and Boeing," in *ISPA/SCEA International Conference*, 2010.

[101] Schrage, D. and Mavris, D., "Integrated product/process design/development (ippd) through robust design simulation-the key for affordable systems," in *Aircraft Engineering, Technology, and Operations Congress*, p. 3892, 1995.

[102] Schuster, P., "Taming combinatorial explosion," *Proceedings of the National Academy of Sciences*, vol. 97, no. 14, pp. 7678–7680, 2000.

[103] Seidl, M., Scholz, M., Huemer, C., and Kappel, G., *UML @ Classroom: An Introduction to Object-Oriented Modeling*. Springer International Publishing, 2015.

[104] Sharma, J. L., *STASE: Set Theory-Influenced Architecture Space Exploration*. PhD thesis, Georgia Institute of Technology, 2014.

[105] Sobek II, D. K. and Liker, J. K., "Another look at how Toyota integrates product development," *Harvard business review*, vol. 76, no. 4, pp. 36–47, 1998.

[106] System, "Merriam-webster online dictionary," June 2016. Retrieved June 20, 2016, from http://www.merriam-webster.com/dictionary/system.

[107] Technology, "Merriam-webster online dictionary," June 2016. Retrieved June 21, 2016, from http://www.merriam-webster.com/dictionary/technology.

[108] Theodore Gray, Nick Mann, M. W., "Online periodic table of elements database," 2013. (http://periodictable.com, Retrieved March 2017).

[109] Ullman, D. G., *The mechanical design process*. McGraw-Hill New York, 3rd ed., 2003.

[110] United States Government Accountability Office, "Defense acquisition: Improvements needed in space systems acquisiton management policy," September 2003.

[111] United States Government Accountability Office, "Space acquisition: Stronger development practices and investment planning needed to address continuing problems," July 2005.

[112] United States Government Accountability Office, "Nasa human space exploration: Delay likely for first exploration mission," April 2017.

[113] U.S. Air Force, "Early systems engineering guidebook," 2009.

[114] U.S. Department of Defense, *Defense Acquisition Guidebook.* Defense Acquistion University, 2013.

[115] U.S. Department of Defense Architecture Framework Working Group, *U.S. Department of Defense Architecture Framework Version 2.0.* U.S. Department of Defense, May 2009.

[116] U.S. House Committee on Commerce, Justice, and Science, "Charting a course: Expert perspectives on NASA's human exploration proposals," February 2016.

[117] Vanderplaats, G. N., *Multidiscipline Design Optimization.* Vanderplaats Research and Development Inc., 1st ed., 2007.

[118] Wertz, J., Everett, D., and Puschell, J., *Space Mission Engineering: The New SMAD.* Microcosm Press, 2011.

[119] Whitehead, J., "Mass breakdown of the Saturn V," in *36th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit*, p. 3141, 2000.

[120] Wiley J. Larson, Ronald W. Humble, G. N. H., *Space Propulsion Analysis and Design.* The McGraw-Hill Companies, Inc., first ed., 1995.

[121] Winn, S. D. and Hamcher, J. W., "NASA/Air Force Cost Model: NAFCOM," 2002.

[122] Zero Point Frontiers, "Beyond LEO architecture sizing tool (BLAST) user guide," 2016.

# INDEX

# VITA

Douglas J. Trent was born on January 6, 1987 in Sacramento, California where he earned the rank of Eagle Scout in 2003, graduated high school from Elk Grove High School in 2005, and went on to attend the California State University, Sacramento. During his undergraduate career, Douglas achieved the Deans Honor list all but two semesters and graduated Magna Cum Laude with a Bachelors in Mechanical Engineering in 2011. He also supported the Associated Students Inc. as an outdoor adventure guide where he was awarded by the university an official commendation for bravery and heroism during his duties. He pursued his postgraduate studies in Aerospace Engineering at the Georgia Institute of Technology, where he joined the Aerospace Systems Design Laboratory (ASDL) under Dr. Dimitri Mavris. During his time with ASDL, Douglas worked on a wide range of projects ranging from orbital debris mitigation, counter-directed energy weapons, the DARPA META program, and space transportation architecture design for NASA Marshall Space Flight Center, earning a Masters of Aerospace Engineering in 2014. While at the Georgia Institute of Technology, Douglas also Joined NASA's Marshall Space Flight Center under the Pathways Intern Employment Program. As a Pathways intern for NASA, Douglas received a range of experience in the fields of environmental control and life support systems, thermal analysis and design, nuclear fuels development, and advanced concepts design.