DISSERTATION

YET ANOTHER MBSE METHODOLOGY (YAMM): A DEVOPS AND EMPIRICALLY-BASED UNIFIED-MODELING, DESIGN, ANALYSIS, AND OPTIMIZATION METHODOLOGY FOR SOFTWARE-CENTRIC DATA SYSTEMS

Submitted by

Thomas M. Booth

Department of Systems Engineering

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 2025

Doctoral Committee:

Advisor: Dr. Sudipto Ghosh

Dr. Daniel Herber
Dr. Nathaniel Blanchard
Dr. Leo Vijayasarathy

ABSTRACT

YET ANOTHER MBSE METHODOLOGY (YAMM):
A DEVOPS AND EMPIRICALLY-BASED UNIFIED-MODELING, DESIGN, ANALYSIS,
AND OPTIMIZATION METHODOLOGY FOR SOFTWARE-CENTRIC DATA SYSTEMS

Data systems consist of a network of communication channels, software that processes, creates, or transmits data across these channels, and the hardware that runs these applications and generates data. Software-centric data systems rely on software to define system behavior, security, and other capabilities. To keep up with rapidly shifting system requirements, many successful organizations utilize software Development and Operations (DevOps) principles to increase software quality and throughput of new software capabilities. Many organizations have modern data systems that include cloud resources and 49% of these organizations lack a Financial Operations (FinOps) team for cloud management and optimization. A third of these organizations spend more than $12 million/year on cloud costs and more than half are regularly 18% over their annual cloud budget. Similar resource management issues exist for aircraft avionics data systems which also cost organizations millions of dollars a year.

Adopting a practical system analysis and optimization methodology is one solution to these organizational budget issues. Model-Based Systems Engineering (MBSE) enables the design, analysis, and optimization of complex systems and has been in practice since the late 20th century. The Systems Modeling Language (SysML) is currently the most common graphical specification language used with MBSE methodologies. Unfortunately, there is insufficient research into MBSE and SysML that show a positive Return On Investment (ROI) for the design, development, sustainment, and optimization of software-centric data systems. Additionally, to keep up with the rapid deployment of software capabilities, a practical MBSE methodology and modeling approach for data systems would need to include DevOps principles.

We present Yet Another MBSE Methodology (YAMM), a novel and practical MBSE methodology towards the design, development, optimization, and sustainment of software-centric data systems. YAMM has refined and extended the Harmony agile MBSE process (Harmony aMBSE) to provide a more prescriptive and tailored methodology to solve data system specific issues. We also present our novel Unified Modeling Approach (UMA) which combines graphical-based system specifications and simulations into a single, empirically-derived system model which accurately predicts data system resource usage, cost, and performance. Together, YAMM and UMA integrate MBSE with DevOps and FinOps principles to improve data system performance and reduce costs using continuous and empirically-driven feedback throughout the system acquisition and sustainment phases.

We demonstrate and evaluate YAMM and UMA using case studies of two different types of data systems that include operationally relevant empirical data. The first case study implemented the YAMM framework to enable the training, selection, and analysis of a sensor fusion Machine Learning (ML) model for a legacy aircraft sensor data system. Results show the value of using compute, memory, and networking resource consumption in addition to performance and accuracy as evaluation criteria for ML model analysis and selection.

YAMM and UMA are demonstrated on the second case study towards the design, development, optimization, and sustainment of a hybrid cloud analytical data system prototype in Amazon Web Service (AWS). The non-optimized prototype system had a quadratically increasing cost with 1 and 5 year cost estimates of $1.9M and $27.7M. Results showed that the unified model had a Root Mean Squared Error (RMSE) of $865 when compared against AWS cost data. The Pareto-optimal design showed a positive YAMM ROI after 361 days with an estimated 5 year savings of $12.7M without reduction in system performance.

Our UMA is limited to systems without physics-based or entity-based simulation requirements. However, results show that implementing YAMM with UMA can reduce costs, increase performance, and provide insights into dominant system characteristics for the design, development, optimization, and sustainment of software-centric data systems.

ACKNOWLEDGEMENTS

In addition to the authors' affiliation with Colorado State University (CSU), we would like to acknowledge funding and support from the US Air Force STEM+M program, the AFSC/SW 309^{th} SWEG, the National Science Foundation (Award Number: OAC 1931363), and Nexus Digital Engineering LLC.

DEDICATION

I would like to dedicate this dissertation to my amazingly patient wife, Dayna, who is and always has been my biggest fan and supporter. Also, to my wonderful kids, Cole, Avery, and Elliott, who were also patient and kept me driven by my desire to make them proud of their dad. I started this during Covid while they were all in elementary and I'm happy to finish before any of them graduate high school.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

The increasing demand on software development across industries to bring new capabilities to market faster than the competition has driven the development and adoption of the Development and Operations (DevOps) methodology. DevOps is defined as a collaboration between software developers and system operators to automate the continuous delivery of new software capabilities which has been shown to increase software quality and the throughput of software capabilities [3]. Additionally, many companies migrate to the cloud to leverage them as a standardized Information Technology (IT) platform for innovating, and optimizing business processes rather than from cost and delivery models [11]. Recent cloud surveys [12–14] show that there is a 21% increase, year over year, in cloud adoption. A third of the companies surveyed spend over $12 million/year on cloud costs. Over half of the organizations surveyed are spending 18% over their predicted cloud budget and 46% report that rightsizing cloud costs against on-prem costs for hybrid cloud systems is one of their biggest challenges. Unfortunately, 49% of these companies lack a Financial Operations (FinOps) team to manage or optimize cloud spending, however 87% of these companies use cost saving as their number one metric for assess their cloud goals in 2025. These surveys indicate that most companies are overspending on cloud resources, each year more companies utilize the cloud, the majority of companies' number one goal is to reduce costs, but roughly half don't have any strategy to fix this issue.

Similar to the cloud resource management and optimization issues, other industries like the aerospace industry must balance their limited resources, costs, and capabilities of their resource-constrained and software-centric data systems onboard aircraft. Aircraft avionics and sensor data systems have limited onboard compute, memory, and networking resources within the Line Replaceable Unit (LRU) and communication buses that connect them. Each software capability for these systems must be designed to balance resource costs with performance [15]. In the race to add warfighter capabilities to aircraft the Department of Defense (DoD) must constantly balance

cost, schedule, and performance. Adding software capabilities that exceed the hardware resources could cost 10's-100's of millions of dollars and delay the schedule. Therefore, balancing software resource usage with performance can save millions of dollars and shorten schedules.

One of the most challenging issues plaguing teams managing system resources is that software performance and resource usage estimation between different hardware is nearly impossible without empirical data. Previous studies have shown it is possible to predict software performance on the same hardware platform [16–18], however without performance and usage predictions for different hardware, applying their work to a large scale system will have limited value. It is more practical to use empirical data to predict the performance across hardware platforms [19]. Therefore any resource management or optimization strategy must be capable of collecting and analyzing empirical data, then using this information towards the estimation and optimization system resource usage and performance.

We define a data system as a network of communication channels, software that processes, creates, or transmits data across these channels, and the hardware running these applications or generating data. Our definition of a data system is a superset of systems known as Information System (IS) or IT systems. We define a software-centric data system as a data system with the majority of its behavior and functionality defined by software. The design, development, and sustainment of software-centric data systems are typically performed using modern software principles and strategies such as Agile, DevOps, and Development, Security, and Operations (DevSecOps). We will use Development, [*], and Operations (Dev*Ops) to denote either DevOps, DevSecOps, or additional automation within these strategies such as safety or other required system certifications. These principles and strategies are modern software development practices [20] based on the concepts of continually and incrementally deploying software capabilities. The challenge with software-centric system is the lack of system-level resource management and optimizations automation which can keep up with rapid software deployments.

Therefore, to keep up with the increased DevOps software capability throughput, an automated and empirically-based resource management and optimization strategy is needed. This strategy

must be able to continually analyze and optimize system performance, resource usage, costs, software capabilities, and stakeholder requirements. A successful strategy meeting these requirements could save organizations millions of dollars a year or more.

For successful strategies meeting these requirements, we look to Systems Engineering (SE) which is a multi-disciplinary field focused on the design, integration, and management of complex systems throughout their life cycle. Furthermore, Model-Based Systems Engineering (MBSE) uses digital engineering practices to facilitate this design, analysis, and optimization for complex systems and has been in use since the late 20th century. Systems Modeling Language (SysML) is currently the most common graphical specification language used with MBSE methodologies and has shown value in its visual representations of complex systems. Therefore, we ask the following questions about the value of applying MBSE to this problem of resource management and optimization of DevOps-based software-centric data systems.

- *What are the key MBSE methodology and modeling attributes that result in a positive Return On Investment (ROI) during the design, development, and sustainment of a software-centric data system?*
- *What other qualitative values does an MBSE methodology and model with these attributes provide?*
- *What is the accuracy and speed of a SysML-based simulation-enabled model that estimates system resource usage and costs as a function of time?*

1.1 System Acquisition and Sustainment

In this section we discuss the acquisition and sustainment phases of a system, their associated timelines and costs, and methods to reduce these costs. The two most costly phases in a system's life cycle are the acquisition and sustainment phases [21]. System acquisition refers to the process of designing, selecting, developing, and deploying a new system. The system sustainment phase focuses on maintaining, supporting, and upgrading a system throughout its operational life. The

sustainment phase is also known as the service life management phase [6] and includes the following areas: Service Life Extension, Modernization and Upgrades, and Disposal and Retirement.

For new systems that are just starting their acquisition phase, Blanchard and Fabrycky [5] identified the need to pull the "System-Specific Knowledge" curve, shown in Figure 1.1, to the left in order to affect system change while it is still relatively easy and incurred costs are lower. This figure shows the relative curves system-specific knowledge, technology commitments, costs incurred, and ease of change for large systems throughout their life cycle. The objective of any quality MBSE methodology focused on system acquisition should be the early identification of emergent system behaviors, cost estimates, or other data that can pull this system-specific knowledge to the left.

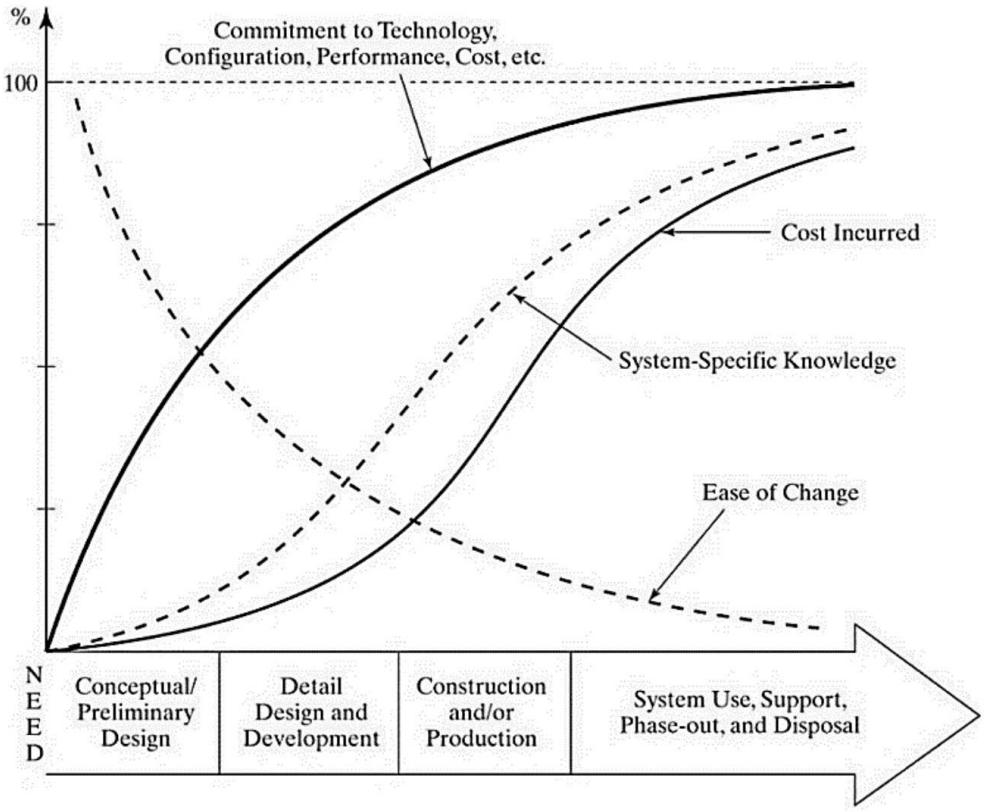


Figure 1.1: Life Cycle Commitment, System-specific Knowledge, and Cost [5]

"Augustine's 16th Law" is an observation from 1979 by Norman Augustine that the defense budget is growing linearly while the per unit cost of aircraft are growing exponentially. His satire

stated that by 2054 the entire DoD's budget will be used to buy a single aircraft. Modern estimates of "Augustine's Law" [22] show the truth behind this observation. The DoD has spent an Order of Magnitude (OOM) more every 20 years on aircraft since this observation. The F-22 and F-35 cost hundreds of millions of dollars per aircraft.

As the aircraft system acquisition costs increase so do the sustainment costs of these increasingly complex systems. DoD sustainment costs are estimated to be anywhere from 55% to 70% of the total cost of the system [21]. This indicates that we should focus on cost savings in sustainment as equally, or more, than the acquisition phase.

The Flexera State of the Cloud reports that cloud spending growth has been roughly 30% year over year [12–14] with growth continuing unabated. Similar to DoD aircraft cost growth, this cloud cost growth is also exponentially increasing. However, this cost increases an OOM every 10 years if this trend continues. This consistent growth does not seem to be tapering off soon [14], as a new era of Artificial Intelligence (AI) and other software services pressure IT professionals to remain competitive in the race to keep up with competition. It appears that this growth represents both new system acquisition and legacy system sustainment modernizations and upgrades.

In summary, the acquisition cost of a system can be greatly influenced in the beginning of the preliminary design phase, so frequent system knowledge feedback to the architects and designers is key to reducing cost. Additionally, the sustainment costs represent the majority of the system cost over its life cycle, so the sustainment phase must continue to receive empirical system feedback to enable continual system optimization and cost reductions. Therefore, a valuable MBSE methodology with positive ROI in system acquisition must also be capable of continual optimizations during the modernizations and upgrades in the sustainment phase. To reduce the labor costs and reach a positive ROI, this methodology needs to automate as much as possible towards producing Pareto-optimal solutions.

1.2 YAMM & UMA

From the previous sections we identified many characteristics of an MBSE methodology which could result in a positive ROI for a software-centric data system over its life cycle. These factors include: early identification of emergent system behaviors, empirically-based cost and resource estimations, automated empirical resource and cost data collection, architecture and design for acquisition, continual system optimization for sustainment, automated model updates, DevOps automated Verification & Validation (V&V) principles, modeling approach and tool with standard Application Programming Interface (API).

There are many valuable MBSE methodologies currently available. However, from our experience in the DoD, most of these methodologies are over-generalized to the point where only MBSE experts understand how to apply them to their system efficiently. Therefore, we looked for a well known agile methodology to refine and extend for our specific purposes.

We chose the Harmony agile MBSE process (Harmony aMBSE) [7] since it is one of the leading agile methodologies [23] with a framework that we could refine and extend to meet our research objectives. The principles of the Harmony aMBSE closely match those of the agile software development philosophy. The Harmony aMBSE philosophy is to iteratively and incrementally design, build, and verify a system using the use case based strategy. This is paired with continual "lessons-learned" feedback to the design stages.

The use case based Harmony aMBSE philosophy is fundamentally different from the traditional SE V-Model life cycle, shown in Figure 1.2. The V-Model assumes most system-specific knowledge is known up front and there is no feedback from the test and verification stages on the right of the V. This is known as a breadth-first design pattern and a top down strategy where each phase in the acquisition life cycle is completed before the next phase is started. The V-model does not work well and is costly for systems with low system-specific knowledge upfront or for modernization and upgrades of systems in sustainment.

Figure 1.3 shows the use case based incremental life cycle of Harmony aMBSE where each circular increment throughout the life cycle designs, develops, and verifies a Minimal Viable Prod-

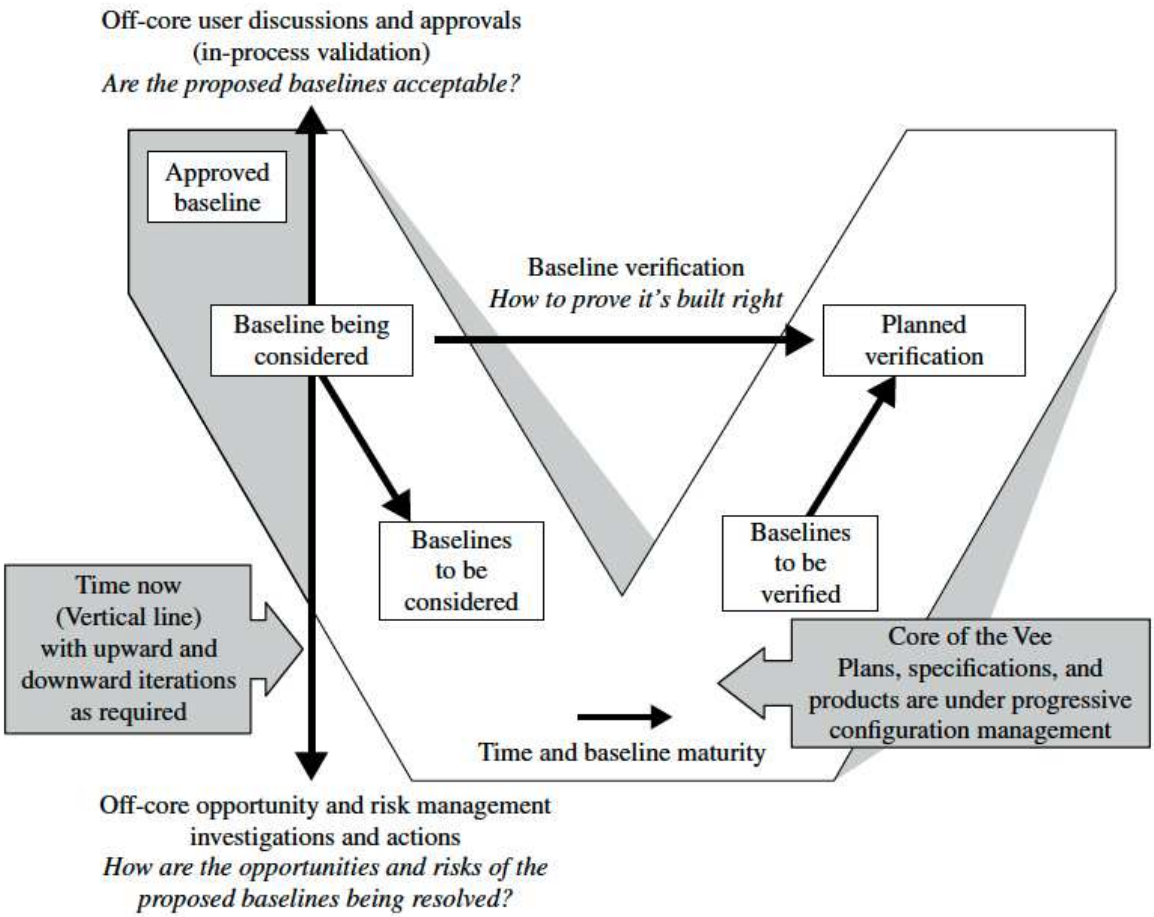


Figure 1.2: SE V-Model Life Cycle [6]

uct (MVP) using a depth first design pattern. Lessons learned from each MVP, with this depth first approach, are fed back into the next increment and use case based MVP. The incremental aspect of this life cycle enables it to pull system-specific knowledge curve to the left. The success of incremental and agile methods in software development is largely due to the ease with which software can be refactored. This incremental and agile framework is the main reason we chose Harmony aMBSE as the foundation for Yet Another MBSE Methodology (YAMM). In addition to this, Harmony aMBSE has the foundational SE aspects of dependability analysis, quality assurance, and project management which are shown at the bottom of this figure.

We draw a similarity between the evolution of software development philosophies and the evolution of our methodology, YAMM, for software-centric data systems. The traditional top-down software development approach evolved into the agile software philosophy of use case based MVPs

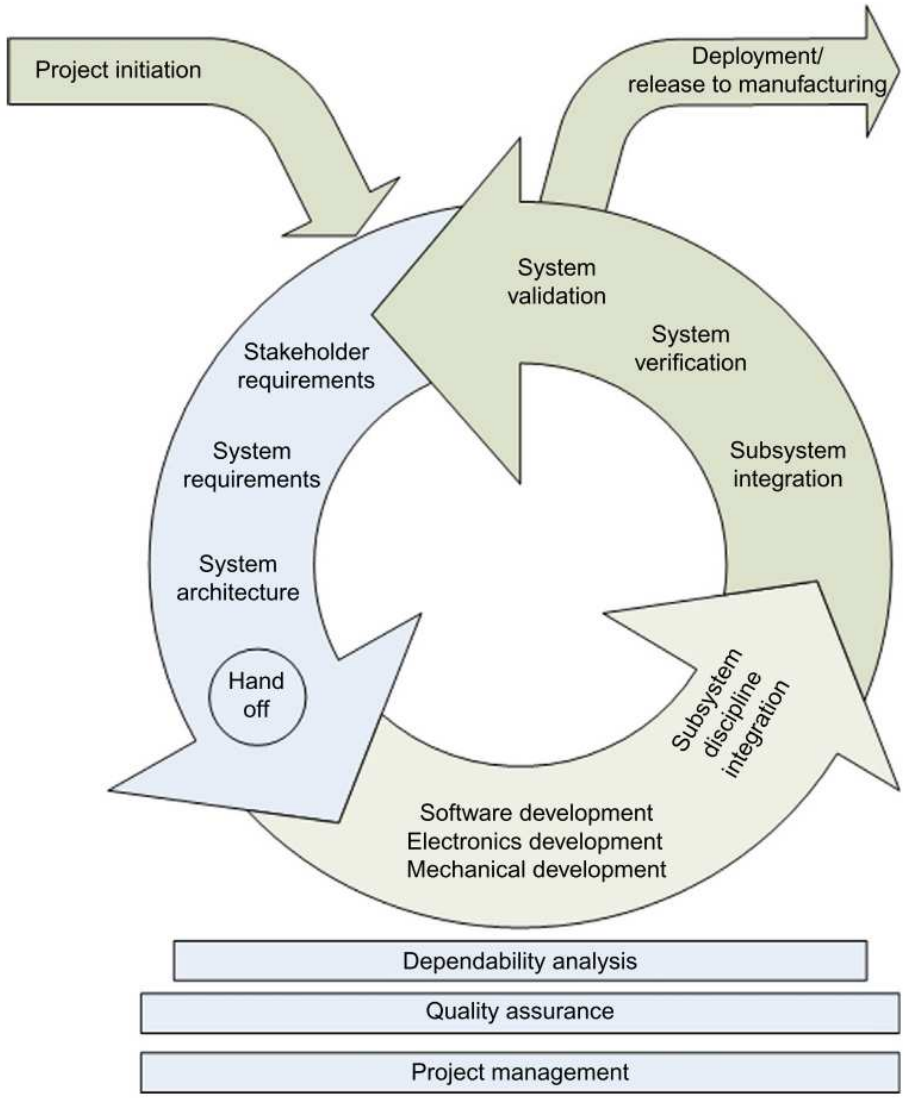


Figure 1.3: Harmony aMBSE Incremental Life Cycle from *Agile Systems Engineering* [7]

and incremental development. Agile was then refined by the DevOps philosophy to automate software development, integration, and deployment through Continuous Integration & Continuous Delivery (CI/CD) and also extended responsibility to the operations team. In this same manner, the top down, breadth first, SE V-model life cycle was adapted by the use case based, depth first, Harmony aMBSE methodology. With YAMM we automated this methodology and extended Harmony aMBSE to include operations. The YAMM life cycle is shown in Figure 1.4.

According to a Forrester study [24], only 46% of companies used or considered using mathematical optimization for tasks like scheduling, sourcing, route planning, resource optimization, etc.

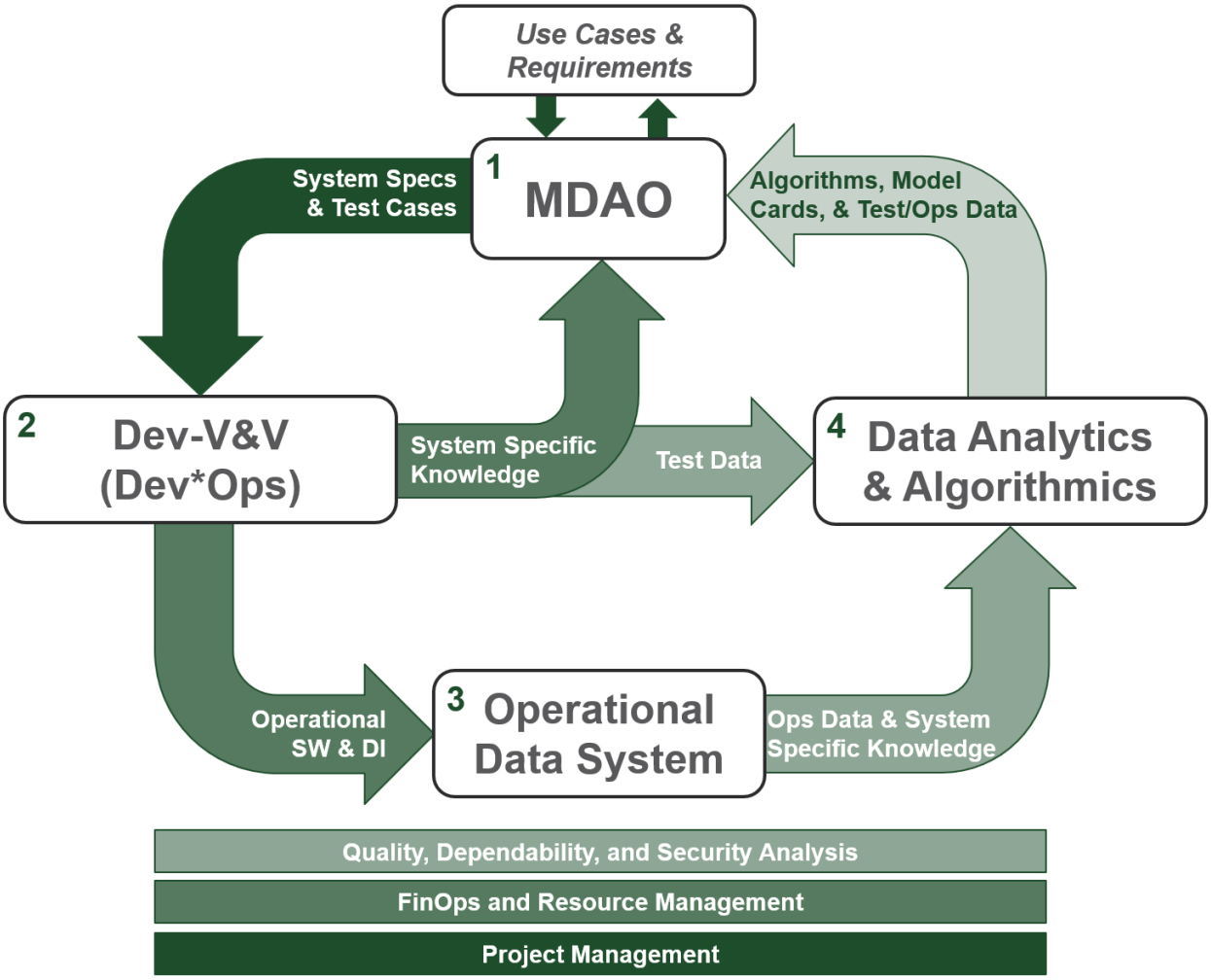


Figure 1.4: YAMM Life Cycle

to reduce costs. Therefore, we enabled YAMM to calculate Pareto-optimal and fuzzy Pareto frontier solutions using simulation matrices or mathematical multi-objective optimization algorithms to identify dominate system characteristics [2]. The insight of these system characteristics can shift the system-specific knowledge curve to the left saving costs and shortening schedules.

YAMM is focused on increasing performance and reducing costs of the system throughout its life cycle and does this through the use of our simulation-enabled Unified Modeling Approach (UMA) [25]. This approach combines Modeling & Simulation (M&S) with system specifications in a single Unified Model (UM). This model is capable of empirical data input for increased simulation accuracy and this empirical data is also used to validate the model and quantify its accuracy.

The use case based simulation within the UM estimates the operational system behavior, performance, and costs well enough for system optimizations. This simulation is a low fidelity Digital Twin of the system in terms of system performance, resources consumed, and costs. This is not to be confused with high-fidelity Digital Twins seen in hard real-time, embedded avionics development capable of simulating systems well enough to test hard real-time embedded running on LRU hardware. The UM enables the system engineers to perform system optimization, analysis, and design through the use of Multidisciplinary Design, Analysis, and Optimization (MDAO) principles. The UMA, system analysis, system optimization steps are represented by the *MDAO* stage in Figure 1.4.

Figure 1.5 illustrates the stages and data flows in the YAMM life cycle that are refinements and extensions of the Harmony aMBSE life cycle. The refinements are colored orange and the extensions are colored teal. Our refinements focus on a practical and more prescriptive approach for software-centric data systems.

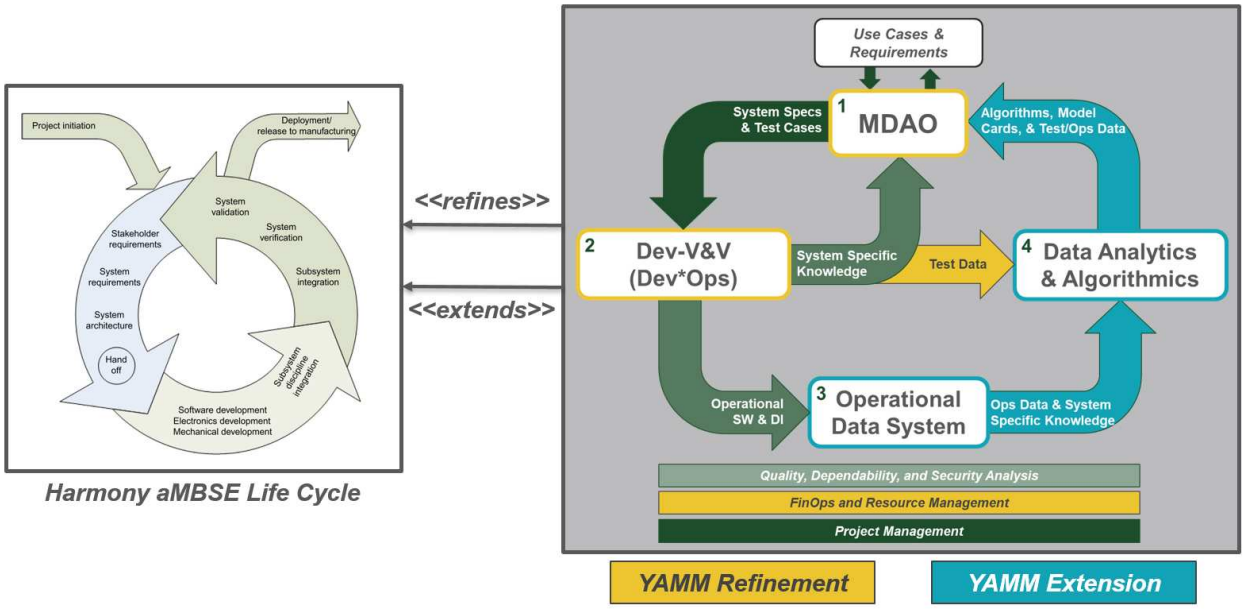


Figure 1.5: YAMM Refinements & Extensions to Harmony aMBSE

The YAMM *MDAO* stage is the first refinement which incrementally and iteratively designs system specifications through Pareto analysis, optimizations, and sensitivity studies enabled by the

UM. This refinement defines our prescriptive process of MDAO techniques for software-centric data systems.

Our second refinement combines software development and system V&V into the single YAMM *Dev-V&V* stage focused on the Dev*Ops CI/CD software development strategy. CI/CD automates the software unit, functional, and system testing, integration, security scans, and any other system requirement before being delivered to the operational environment. The validation in this stage is completed by users, early adopters, or dedicated stakeholder testers of the system. For the DoD, these dedicated validation testers are known as the Operational Test (OT) community. Test data from the V&V activities in this stage is used as empirical resource usage and cost feedback.

The last YAMM refinement specifies a FinOps and resource management focus to the foundational SE aspects at the bottom of Figure 1.5. These are enabled by the refinements and extensions already, but are foundational to software-centric data systems.

The *Operational Data System* stage shown in Figure 1.5 is the first YAMM extension. Collecting operational resource usage data and metadata automatically is its primary focus. This data is sent to the next YAMM stage for analysis. The secondary focus of this stage is to extend responsibility of the design to the operations team in a similar fashion as DevOps.

The final extension is the *Data Analytics & Algorithmics* stage which is a crucial step for the feedback of empirical data to the *MDAO* stage. This *Data Analytics & Algorithmics* stage is responsible for the analysis of operational and system test data, creation of meta algorithms to integrate into the UM, creation of Machine Learning (ML) Model Cards, and the formatting of empirically derived UM validation data. The data and algorithms that are passed from the *Data Analytics & Algorithmics* stage to the *MDAO* stage are the primary sources of capturing the system's behavior, resource usage, and system costs. These sources are integrated into the UM or used as sources to update the model's system accuracy as the system changes.

Aside from empirical data feedback, YAMM enables tighter collaboration between system engineers, software developers, and operations to increase system-specific knowledge early in the life cycle. Data system architecture, MBSE, software development, operations, and cloud archi-

tecture knowledge is currently so deep and broad that it takes several teams to keep up with the latest technologies in these disciplines. Therefore, tight collaboration between system engineers, software developers, and ops teams is vital to pull the system-specific knowledge curve to the left.

In Chapter 2 we discuss the current advantages and disadvantages of several different MBSE methodologies other than Harmony aMBSE, different M&S approaches, previous work done with collecting empirical data, numerical methods and analysis techniques, and system optimization approaches.

Chapter 3 defines each stage of the YAMM life cycle and its data flows in more detail. This chapter also defines the UMA in terms of its processes and modeling structure that enables modular models and creates a set of reusable modeling libraries.

1.3 Case Studies

The first case study is focused on the aerospace industry and demonstrates YAMM for the design and system analysis involved with the update of an ML-based sensor fusion algorithm on a notional legacy aircraft using open National Aeronautics and Space Administration (NASA) avionics sensor data. We demonstrate our novel ML algorithm selection and system analysis approach enabled by the operational data produced by the NASA aircraft data. Our approach is a systematic and iterative methodology for selecting a set of optimal ML techniques while tuning the ML hyperparameter settings simultaneously. We also built a novel extension to ML Model Cards which enables a system analysis process that selects the best ML approach based on resource usage, accuracy, and performance, rather than accuracy alone.

The second case study is focused on the design, development, deployment, and optimization of a new hybrid-cloud data system. This case study demonstrates a notional data system acquisition that transitions into its sustainment phase. This cloud data system represents a big data system capable of processing and analyzing test and operational data from aircraft or other business systems. We implemented YAMM and our novel UMA for the design, development, deployment, and optimization of a hybrid-cloud prototype data system deployed in Amazon Web Service (AWS).

This YAMM implementation and research took over 4 YAMM iterations, 5 UM versions, and 2 optimization and analysis cycles over 21 months. Our results show that we captured emergent system behavior and accurately estimated data system costs and resource usages for use in an optimization algorithm. The model cost predictions accuracy had an Root Mean Squared Error (RMSE) of $865 compared to the prototype. We showed that the implementation of YAMM had a positive ROI and had a predicted cloud cost savings of $12.7M over a 5 year period compared to the AWS prototype.

The main objectives for implementing YAMM on the hybrid-cloud data system were to reduce costs, increase performance, and develop our cloud governance policies. We developed the following specific goals and successfully accomplished them during the case study:

1 Identify the best recording format and communication protocols between three example aircraft formats.

2 Identify the optimal Extract, Transform, and Load (ETL) pipeline storage utilization strategy between Elastic Block Storage (EBS) and Amazon Simple Storage Service (S3) for the three formats.

3 Define the optimal data retention and data archive policies.

4 Estimate on-prem Capital Expense (CapEx) and Operating Expense (OpEx) costs for the hybrid cloud system.

Chapters 4 and 5 describe these case studies and step through the application of YAMM in more detail. Demonstrating our methodology on these relevant data system case studies enables the measurement of quantitative values and examples of the qualitative values of YAMM and the UMA.

1.4 Contributions

Our MBSE methodology, YAMM, and our UMA [25] are the main novel contributions we're providing to the field of system engineering. Specifically, the novel YAMM refinements and extensions of Harmony aMBSE which we paired with our UMA. Together these novel contributions include the following lists of MBSE methodology and modeling approach attributes focused on

generating a positive ROI for the design, development, sustainment, and optimization of software-centric data systems: **YAMM Methodology Attributes**

1 Collect operational and system test resource usage and performance data towards validation and increased accuracy of system models and system analyses.

2 Perform system optimization, Pareto and fuzzy Pareto studies, resource analyses, and sensitivity analyses.

3 Provide continuous feedback of system specific knowledge to optimize the design and operations throughout the system's life cycle.

UMA Modeling Attributes:

1 Create semantically consistent system specification models to increase model reliability

2 Visualize complex system models with intuitive, graphical representations to increase human interpretation accuracy.

3 Merge specifications and simulations into a single, unified model to ensure consistency between specification and simulation behaviors.

4 Accurately capture emergent system behaviors using time-dependent simulation.

5 Utilize modular and reusable model libraries to reduce modeling and design time.

6 Automatically verify MBSE model functional specifications using simulation.

7 Automate validation of the MBSE model specifications, simulations, and tool set.

8 Standardize an open model simulation interface to integrate with standard data analytics environments.

Our other novel contributions as a result of our research include: 1) an automatable ML Operations (MLOps) technique for resource-limited systems which analyzes and selects ML models based on resource usage, performance and accuracy [26], 2) ML Model Card extensions which include the ML model's resource usage, accuracy, and performance that enables the automation of our novel MLOps technique [15], 3) the specification and integration of SysML-based system models to a common analysis environment which also enables the automation of our novel MLOps technique [15].

1.5 Conclusions

Results show that our novel UMA is capable of creating and validating accurate, time-dependent, and simulation-enabled SysML models which are capable of capturing emergent system behaviors, estimating data system resources, costs, and performance. These models can integrate with common analysis tools, like python, to provide multi-objective optimization and Pareto analysis capabilities which can identify dominant system characteristics which can lead to more efficient and robust system designs.

Our UMA also produces a modular unified model with a reusable set of modeling libraries to enable faster scalability for the size and complexity of models which can save engineering teams time and money. With only one model for simulation and specification, our unified models reduces: integration costs, additional tool license costs, version control tracking across models, and errors with using the wrong Authoritative Source of Truth (ASOT).

Our results also show that YAMM can affect system change early in the acquisition phase while it is still relatively easy and incurred costs are lower. We do this by providing a continuous feedback of knowledge which, in reference to the "System-Specific Knowledge" curve in Figure 1.1, shifts system knowledge to the left and reduces costly system refactoring.

YAMM's continual empirically-derived updates to the UM ensures that the system model remains accurate throughout the life cycle of the data system. Keeping a model updated and accurate after initial acquisition ensures it can be used for the design, analysis, and optimization of system sustainment updates and modernizations which can save design time and reduce system costs. Additionally, with continually updated and accurate cost and resource estimations, YAMM can be implemented in place of an organization's FinOps method which reduces time spent on resource management and optimization activities.

Using SysML or another graphical and semantically consistent system specification language ensures the model is correct, comprehensive, and unambiguous which can result in system specifications that are more likely to be understood by stakeholders which can result in fewer issues with system integration, updates, and modernizations which also saves time and money.

YAMM's prescriptive nature for data systems better enable non-MBSE experts the ability to design and optimize systems by minimizing the requirement to refine and extend a more general SE methodology. This prescriptiveness also reduces the work required of experts which can save time and money.

A fundamental limitation of our UMA stems from our assumption that the system simulation can be done completely inside the unified model. This limits the scope of the UMA to systems without physics-based or entity-based simulations and analyses which are usually done in specialized tools outside the system model. We concluded that the benefits and widespread need for a practical and more prescriptive data system modeling and MDAO methodology outweighed the benefits of a methodology with broad applicability to many other types of systems.

Another limitation to the UMA is the current lack of production-ready tools capable of high performance simulation which would result in timely system optimizations. Current production tools are inefficient for this task and use large amounts of compute and memory which increases simulation run time and costs. The new SysML version 2 language and currently emerging set of supporting tools show a potential for solving this limitation, however these tools have not yet been proven for production. A work around to this limitation is to increase the size of compute resources and parallel jobs running these simulations in current tools to reduce simulation run time.

In conclusion, YAMM is a practical and prescriptive MBSE methodology which enables the design, analysis, and optimization of software-centric data systems. YAMM and the UMA successfully adopt modern software DevOps principles and applies them to the MBSE, MDAO, and FinOps disciplines to reduce costs and balance performance and other organizational objectives. Results show that YAMM is equally capable for system acquisition as it is for system sustainment updates and modernizations. Our results also show that our novel YAMM refinements and extensions to Harmony aMBSE for software-centric data system can save organizations millions of dollars per year. Additionally, results showed that, when paired, YAMM and the UMA can save cloud-based companies millions of dollars a year with a positive ROI possible within the first year. These results show that YAMM also generalizes well to different types of data systems such

as resource-limited avionics sensor data systems and hybrid cloud data processing and analytical data systems. We provide additional details about the contributions and conclusions in Chapter 6. We also summarize the results and conclusions of both case studies, fully answer our research questions, provide the challenges and limitation of YAMM and the UMA, and list future research.

Chapter 2

Related Work

We break this chapter into four sections; 1) data and information systems, 2) MBSE methodologies, 3) system modeling, analysis and optimization, and 4) data collection for use in data analytics and producing algorithms and meta algorithms. Section 2.1 reviews the background of data categorization within data systems such as operational and analytical data. It also provides the background of information system categories that align with the processing of operational and analytical data.

Section 2.2 provides a background into an MBSE methodology evaluation framework, several well known methodologies, agile principles, DevOps and DevSecOps software development philosophies, and agile systems. This is not a comprehensive review of all MBSE methodologies, rather it is a summary of valuable methodology attributes and what is lacking in methodologies towards the practical application of data system design, sustainment, and optimization.

Section 2.3 discusses research that is focused on M&S of complex systems, which includes traditional MBSE techniques as well as non-traditional MBSE techniques towards simulation and estimation of complex systems. Specifically, this section consists of cloud-based application performance estimation software, MBSE modeling approaches, and M&S using traditional executable MBSE languages such as the SysML and the Foundational Unified Modeling Language (fUML). We also review previous work in mathematical optimization techniques for complex systems, Systems of Systems (SoS), and architectures.

Section 2.4 provides a brief overview of previous work done collecting the statistically significant amount of data for numerical modeling behaviors in a cloud environment. We then delve into different numerical methods that pertain to modeling data for the purpose of capturing data system behavior in a system model. We review discretization, parameterization, and ML training and selection techniques. We also focus on the background of ML and discuss techniques involved in ML hyperparameter tuning, ML towards sensor fusion, and ML Model Cards.

2.1 Data and Information Systems

This section provides an overview of the different categories of data and data systems. In most data systems there are a variety of data that can be sorted into the following five data categories:

- **Operational data** is used in the real-time operation of a data system. Examples include purchase transactions, current inventory, real time sensor output, or avionics buses.

- **Test data** is used during the testing phase of an operational system to verify and validate that it meets requirements and users' needs. The examples are the same as those for operational data, but the data system is not operating in the operational environment. In aircraft data systems, test data includes Developmental Test (DT) data and OT data with the differences focusing on system verification vs validation, respectively.

- **Analytical data** is the aggregation of operational data used to capture trends, gain insight into the organization's operations, or predict future maintenance activities. Examples of analytical data are sales history, customer demographics, historical maintenance records, or historical sensor data.

- **Resource data** is information about the data system's resources consumed during operation. Examples are the compute, storage, and network resources used throughout the day, month, or year. Consumption of resources in the cloud, for instance, are charged at rates of seconds, minutes, hours, or months. Cloud costs are also considered resource data since they are a function of the resources consumed and cloud services such as Software as a Service (SaaS), Platform as a Service (PaaS), AI, and many others.

- **System metadata** is information about the system and the data itself. This includes the system name, sub-system names, resource types, data formats, file sizes, recording formats, or data sources.

There are many types of data systems. Some examples include the internet, business information systems, or aircraft avionics systems. These system can be categorized into operational

systems, analytical systems, or a combination of the two. For instance, the Online Transaction Processing (OLTP) system category is defined as a system used in transactional applications, process user requests, respond in real time, and primarily use operational data.

In contrast, the Online Analytical Processing (OLAP) system category focuses on data analysis which is comprised mainly of analytical data. An example of an OLAP system is the Massively Parallel Processing (MPP) OLAP system frameworks like, Mapreduce and Hadoop, which paved the way for a new generation of analytic data systems that could process multi-terabyte data sets in parallel on large clusters [27]. However, these MPP OLAP systems operated mostly on analytical data, which was historically considered data at rest as opposed to data that is rapidly updated with the latest operational data [27]. We categorize data systems such as aircraft and their operational network into the OLTP category, while the data system that aggregates aircraft avionics or sensor data for ML training or trend analysis would fall into the OLAP category.

As organizations became more agile, they demanded real time analytical insights into their operational data within their OLTP system. This necessitated the evolution of information systems that combine the OLTP and OLAP environments, which led to the idea of the Hybrid Transactional/Analytical Processing (HTAP) [28] system category, where operational data gets continuously ingested and serves up real time data analytics. Bohm et al. [27] refers to these HTAP systems as operational analytics systems. The DoD is working on leveraging real time analytics of operational data as a strategic asset [29, 30] in their aircraft and networks of information systems. An example of one of the most recently developed operational analytics system architectures is the Data Mesh developed by Dehghani [31].

Whichever category of data system is being developed, test and resource data and system metadata need be collected and analyzed to enable continual data system optimization as data systems change throughout their life cycles.

2.2 MBSE

In the fields of data and information systems, the lack of MBSE experience can impede the use of disciplined system design and analysis. Further complicating the adoption of MBSE for practitioners in these fields is the evaluation of which MBSE methodology is best suited for their application. Weilkiens [1] recommends data system organizations use a structured framework to evaluate the many MBSE methodologies [32] available. An MBSE methodology evaluation framework is outlined briefly in Section 2.2.1. We review several well known methodologies in Section 2.2.2 and then review agile systems engineering in Section 2.2.3.

2.2.1 MBSE Methodology Evaluation Framework

The Framework for the Evaluation of MBSE Methodologies for Practioners (FEMMP) is one example of an evaluation framework that lists a wide range of criteria to consider [1] when choosing an MBSE methodology. Tables 2.1, 2.2, 2.3, and 2.4 define the FEMMP catalog of criteria against which MBSE methodologies can be assessed. These tables were pulled from the Weilkiens paper [1], simplified, and ordered by category.

Table 2.1: FEMMP Evaluation Criteria [1]

Title	Description	Category
Learning Curve	How steep does the learning curve feel?	General
Suitability for Beginners	How easy to understand is the methodology for MBSE novices?	General
Training Effort for SEs	How much training does it take for experienced SEs to implement the methodology correctly?	General
Industry Domains	Which industry domains does the methodology support particularly well has it been developed for?	General
Creativity Support	How does the methodology foster a creative environment for the systems engineering team?	General
Support of Other Standards	Does the methodology also support other standards or norms? If so, how well?	General

Table 2.2: FEMMP Evaluation Criteria Continued [1]

Title	Description	Category
Tailoring	How easy is it to tailor the methodology (add, delete or change processes or process steps, object definitions or toggle tool features on and off)?	General
Complexity	How often is the methodology 'interrupted' by relying on external processes and/or non integrated tools, e.g. by a 'paper-review'?	General
Simulation	How well does the methodology provide for an integrated simulation of the various model abstractions?	General
Scope	For what engineering purpose is the methodology , suited(innovation, improved products refactoring, reverse engineering, etc)?	General
Documentation	How well is the methodology supported (books, manuals, case studies, on-line help community, websites, interactive support, user feedback etc.)?	General
Training	How well is training supported (availability, consultants, coaches, e-training, background knowledge required)?	General
Process Info Capture	How well does the tool capture the information generated throughout the process?	Info
MBSE-SE Info Exchange	How well does the tool facilitate the exchange of the information between the MBSE domain and the non-MBSE tools	Info
Philosophy	Are Model Elements clearly distinguished from Diagram Elements (separation of content from representation)?	Language
Language	What Modelling Language is used (If NOT SysML: How well does it define the real-world semantics unambiguous, of the engineering, are elements strictly typed, is their meaning do they have a defined purpose etc.)?	Language
Integration	How well can the model be integrated with specialty engineering models	Language
Modeling Features	What modelling features and approaches are included from the reference framework ISO 42010 (Architecture)	Model
Modeling ISO 15288	How well does the MBSE methodology support modelling of the ISO 15288 processes?	Model
Abstraction	How well does the model support keeping it abstract through multiple levels?	Model

Table 2.3: FEMMP Evaluation Criteria Continued [1]

Title	Description	Category
Meta-Model	Does the methodology provide a (semantic) meta-model, ontology or similar?	Model
Scalability	How well does the model scale (suitable for large projects, 'grows' with time without becoming cumbersome, does it require partitioning e.g. in a tree)?	Model
Variants	How well does the methodology support the variant management?	Model
Independent View Generation	How well does the methodology support the generation of views that can be read in another MBSE language?	Model
Redundancy	How well does the methodology prevent duplication (of work, model elements, artifacts communications and reports)?	Model
ISO Standard	What process steps of ISO 15288 are covered?	Process
Framework	What views from the reference framework are used (MODAF, DODAF...)?	Process
Consistency	Is the process self-contained (are in-/outputs to all steps connected)?	Process
Precision	How precisely is the process implemented in terms of semantics and sequence, i.e. is it strongly enforced, or can 'wildcard' be used as 'work arounds' that would reduce the model quality?	Tool
Information Security	How secure is the data/information exchange between parties	Tool
Perspectives	To what level is the creation of experts' perspectives automated	Tool
Reporting	How quickly are standard/custom reports, is design documentation created?	Tool
Admin	How well does the tool help to minimize work that isn't creating any value	Tool
Checking	Does the tool support consistency checking of the model? (Automated detection of wrong content and/or formats, flagging of 'loose ends' etc)?	Tool
Navigation	How easy is it to find the correct model element (are elements links, users guided in the process, information well)? aggregated, need to 'jump' screens	Tool
Intuition	How intuitive is the tool to work with (compliance with UX conventions, standard tool reactions e.g. tool tips, double/right click, drag drop, delete, Keyboard shortcuts,)? spell check, familiar operations e.g. as MS-Office	Tool

Table 2.4: FEMMP Evaluation Criteria Continued [1]

Title	Description	Category
View	How easy is it to configure the UX dynamically (define a matrix with sorting filtering of columns and rows,)? store customized view, annotation, comment	Tool
UI	How readable is the UI (good use of screen estate and colour, zoom, can fonts and sizes be changed, is information well presented?	Tool
Multi-User Environment	How well does the tool support the distribution of the work among different parties	Tool
Connectivity	How easily can the information be exchanged with other tools (what standard API are provided by the tool, what API can be added, Is import/export open protocols guided by wizard, can it be rolled back, what the quality control mechanism etc.)?	Tool
Reuseablity	Does the tool allow to reuse any type of Modelling Element across projects	Tool
Support	How well is the tool supported (vendor response times, 24/7 helpline etc.)?	Tool

In the search for an MBSE methodololgy, Weilkiens has noted that rather than finding a tool to use with an MBSE methodology, users, in practice, often select the tool to then adapt the (arguably more important) methodology to it [1]. This can cause difficulties for new organizations to extract value from their methodology. The FEMMP process also provides the following standard steps for evaluating MBSE methodologies (list extracted from Weilkens [1]):

1 *Methodology:* An overview in free text format with illustrations and references for more information.

2 *Highlights:* Selected features making the methodology unique or interesting are explained and discussed in detail.

3 *Case Study:* A brief summary of notable points from applying the methodology to a standard challenge including some of the artifacts, design solutions and/or reports.

4 *Evaluation:* A table listing the evaluation of the framework criteria with answers, explanations, and comments.

5 *Discussion:* A summary of the process and its results.

2.2.2 MBSE Methodologies

The Executable System Engineering Method (ESEM) by Karban et al. [33] is based on the top-down system and functional decomposition of the Object Oriented Systems Engineering Method (OOSEM) [34]. OOSEM is applicable for many generalized purposes and ESEM is capable of simulation, however we strive for an Agile approach to systems engineering and software development and a top down approach is not ideal for this approach.

The Model-Based Systems Engineering Process (MBSAP) [35] is also a top-down system decomposition, however it improves on top-down by focusing simulation at each level of system abstraction (i.e. operational, logical, and physical) with a feedback loop built to continually update the design of the system. The MBSAP also applies the concepts of Object-Oriented Design (OOD) in a similar manner to OOSEM and ESEM, which is a common and well thought out concept for MBSE methodologies. While this could be used as an agile methodology we are looking for one developed with agile software development practices as their core tenet.

Model Development Operations (ModDevOps) [36] is an MBSE methodology that merges the philosophies of MBSE and DevOps. This methodology is a systems/software co-engineering culture and practice that aims at unifying systems engineering Models (Mod), software development (Dev), and software operation (Ops). The main characteristic of ModDevOps is to strongly advocate abstraction, automation, and monitoring at all steps of system construction, from integration, testing, releasing to deployment and infrastructure management. Other than the website we could not find many more details to fully understand their methodology. From what we could find about it, this methodology relies heavily on automated code generation from SysML and other modeling languages like Unified Modeling Language (UML) and the Architecture Analysis and Design Language (AADL). This does not account for large data systems that primarily use common Open Source Software (OSS) for their data system services. Unfortunately, this methodology has been refined and extended into a discipline does not fit our needs.

Design methodologies have been developed for automated manufacturing system software design [37] as well as the automobile's Real-Time Operating System (RTOS) software design [38]. These methodologies also use automatic code generation from models and perform rapid prototyping to ensure code works on hardware. However, these methodologies fail to analyze resources and costs of their data systems. We are unsure if companies perform these analyses, in any case we have not found an MBSE methodology that explicitly focuses on these analyses with feedback into a system optimization technique.

Several recent MBSE publications [39–41] focus on DoD acquisitions of new system and do not address the method of keeping the model up to date during the more expensive and longer sustainment phase [21]. Our own observations within the DoD show this same lack of MBSE support for software upgrades throughout system sustainment. This indicates a lack of MBSE design, analysis, and optimization support for new software capabilities after the initial system acquisition.

2.2.3 Agile in Engineering

Douglass, in his Agile System Engineering [7] book, adapted the agile software development mindset and core values to the field of SE and more importantly MBSE, which enables agile SE. Douglass describes the integration of systems engineering, SysML, MBSE, and agile software development practices into a cohesive, usable approach for systems engineering. The approach in this book is based on the Harmony aMBSE [7, 42–44]. The Harmony aMBSE is an agile, model-centric approach to developing the engineering data required of systems engineering and utilizes use cases as its method of refining specific areas of the system at a time. This methodology also employs OOD and iteratively executes this use case based model and verifies it against design synthesis data as a method to continually improve the system specifications and simulations. Ideally, this method is utilized throughout the life of the system to include design, initial development, and continuous development and sustainment.

Agile software development has been adopted and practiced in the software IT community sometime after the Agile Manifesto [45] was developed by the Agile Alliance [46] and released in 2001. The manifesto contains the following four core values, which capture the intent of breaking from the standard and inflexible practices that were deemed to be focused on the wrong values of software development.

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

The DevOps philosophy is the natural progression of agile software development. DevOps is defined as a collaborative and multidisciplinary effort within an organization to automate continuous delivery of new software versions to an operational environment, while guaranteeing their correctness and reliability [47]. A DevOps survey by Leite et al. [47] indicates increases in software production throughput and quality when implementing a DevOps strategy for software development. Similarly, DevSecOps philosophy [3] is a DevOps philosophy that includes security measures within the continuous automation involved with deploying new software versions to operational environments. We will use Dev*Ops to denote either DevOps, DevSecOps, or additional automation added to these strategies.

To quote from Douglass [7]: "The Harmony Agile Systems Engineering process is an agile, model-centric approach to developing the engineering data required of systems engineering; requirements, architectures, interfaces, and dependability analyses are the foremost among these. The Harmony process has been developed and honed by the decades of systems experience of the author in real projects that fly, drive, and otherwise perform all around the world."

Aside from Harmony aMBSE, another aspect of SE looks at creating agile systems. This is referred to as **agile systems** engineering rather than agile **systems engineering** by Haberfellner and de Weck [48]. They draw the distinction between agility in SE versus agility in the resulting system itself. In the first case, the emphasis is on exploring the design space and delaying the requirements

freeze as long as possible to better understand and adapt the system specific knowledge or changing requirements before committing all your resources. The second case is relevant to systems that can respond to changes in requirements after initial fielding.

One example of an agile system is the Data Mesh architecture [31] which decentralizes data warehouses and data lakes. Each instance of these are required to provide data as easily and painlessly as possible to its data consumers and other data warehouses. The Data Mesh architecture refers to this as providing data as a product. The decentralization of these Data Products and underlying self-service platform enable this architecture to react to changes quickly.

2.3 System Modeling, Design, Analysis, & Optimization

This section summarizes previous work and historical issues related to the estimation of software performance and cloud cost estimations. Previous work includes MBSE-based simulations and non-MBSE M&S solutions to these problems. In addition to modeling and design capabilities, this section also summarizes different techniques for system analysis which include Pareto and fuzzy Pareto analyses used to identify dominate system characteristics. Techniques for system and SoS optimizations are covered as well as algorithms and processes for implementing single objective optimization and Multi-Objective Optimization (MOO).

2.3.1 Modeling and Design

This section covers previous M&S work towards building a better understanding of complex systems. Section 2.3.1.1 describes previous work done towards cloud performance and cost M&S with non-MBSE based methods. Rather than traditional MBSE techniques, this section reviews software geared towards the context of FinOps [49] and estimating currently developed cloud-based application performance and costs. Section 2.3.1.3 reviews previous work in executable MBSE-based models to simulate complex systems towards a better understanding of complex and time-dependent system behavior.

2.3.1.1 Cloud Performance and Costs

A subset within the domain of complex software-centric information systems is the cloud-based information system. Many cloud-based organizations have come to the conclusion that they need to analyze how to best design/re-design their cloud-based information systems to reduce costs while keeping their current capabilities. Recently many companies are reducing use of the cloud for their information systems [13] in favor of building on-prem or cloud hybrid solutions. According to the State of the Cloud Report [12], the number one challenge in 2023 was managing cloud spending. It is reported that 66% of the 753 executives and cloud decision-makers surveyed said that that they were over their cloud budget by 18% for 2022 with an anticipated 30% growth in cloud spend in 2023. These organizations self-estimated that they wasted 28% of their cloud budget. In 2024, the number two cloud challenge was the assessment of on-prem vs cloud costs and rightsizing the best combination [13]. Accurate estimates of software performance, behavior, and resource usage on-prem and in the cloud are required when trying to rightsize the best hybrid combination and estimating software performance and costs across a range of different hardware platforms is difficult. While previous studies have shown it is possible to predict software performance to the same hardware platform [16–18], it is still difficult to apply this to a large scale hybrid system with many software services deployed. However, it has been shown that using empirical data to predict the performance across hardware platforms is possible [19], therefore it is critical that any design tool used to rightsize information systems must be capable of ingesting empirical data to enable large scale system resource usage.

Cloud cost estimation is not as simple as hardware estimation for an on-prem data system. For instance, an on-prem data system has fixed hardware costs, facility usage, and power cost estimations for Total Cost of Ownership (TCO) [50–53]. However, cloud costs are difficult to estimate because the cost is a function of cloud usage, therefore costs can fluctuate significantly. For instance, cloud computing is capable of scaling resources up or down based on high demand and fluctuating work loads. In this instance, the company is ideally only charged for the scaled resources during peak need, then charged less when need is reduced. This cloud-based business

model makes sense if the larger resources are only occasionally required and purchasing an on-prem solution of the same resource size would largely go unused. In other words, this cloud-based business model works well when there are only occasional peaks in demand and not a constant demand for resources. These cost fluctuations are the source of budget over-runs and the ability to estimate the performance and usage of cloud resources is difficult.

2.3.1.2 Cloud Modeling and Estimation Tools

This section describes the previous work done towards cloud performance and cost M&S with non-MBSE based methods. Developing M&S capabilities from scratch using a common software language is the way all large commercial M&S tools started. We could develop our own M&S and ensure we have all the capabilities we need in a simulation, however this is extremely time consuming and costly. There are many open source projects we could leverage instead.

Salindeho et al. [54] developed a cost-benefit analysis comparing TCO for on-prem digital infrastructure costs associated with using either of the two main Cloud Service Provider (CSP) (AWS and Azure) over a 5 year span for a universities network use case. In this instance, the CSP costs were cheaper, however the authors used the CSPs own cloud estimation tools which don't account for performance of software on the CSP's Virtual Machines (VMs) to process data of different types of formats. Nor does this include the cost of the internet, any on-prem hardware needed to access the cloud, or estimations of system emergent behavior. This may be a valid estimation tool upfront, but quickly becomes less accurate as empirical data is collected and new estimates are calculated.

Truong and Dustdar [55] developed a composable cost estimation application of cloud compute for a next generation sequencing analysis workflow use case which is similar to our data system use cases. They account for 11 different cost models or estimates for cloud resources and costs. These span from storage, to VM, to data transfer in and out of the cloud, and parallel work flows. Their first four basic cost models are those that are provided by the CSP estimation tools, however the other seven cost models account for differences not captured by these CSP estimation tools and provide more accuracy in cloud cost estimations.

Cho and Bahn [56] created a cost estimation model for cloud Infrastructure as a Service (IaaS) that is 99.3% accurate in comparison with the actual cloud charges. However, this cost estimation is a reactive rather than a predictive estimation model. This model deploys a monitoring daemon to the current cloud system to measure usage then calculates a cost estimation based on these measurements. This is not a model that can make predictions of the cost and performance a-priori to inform design and architectural decisions. However, this tool would be useful for collecting data to feed into a numerical model of the IaaS for a predictive system methodology.

CloudSim [57, 58] is a framework for modeling and simulation of cloud computing IaaS and SaaS. This M&S toolkit evaluates the performance of cloud provisioning policies, application workload models, and requirements. Additionally, CloudAnalyst [57] is a CloudSim-based analysis tools to help application developers evaluate the best Cloud provider for their services based on differing pricing models, geographical regions, and data center locations. NetworkCloudSim [59] extends CloudSim's capabilities to include scalable network and generalized application models for more accurate evaluation of scheduling and resource provisioning policies. CloudSim and its associated extensions are partially reactive with some predictive cloud performance estimation. CloudSim produces performance estimates based on software that has already been written for a known cloud-based system. While this is a useful tool that can be used by the system engineer of a legacy system, a full system architectural analysis and design capability is needed for new systems early in the design process. However, system engineers hoping to capture the behavior of a currently built system will need to learn the complexities of this toolkit to capture their system's behaviors. While this is a high quality and accurate cloud resource modeling tool and may provide valuable prototype information and initial data collection, the disciplined system design and analysis must be accomplished for design, analysis, and optimization capabilities in separate tool increasing integration work. Furthermore, this tool would less useful during system sustainment, after software is deployed to the operational environment and data collection has started.

Recurring cloud costs, on-prem recurring costs, and upfront on-prem costs need to be simulated to right-size a hybrid cloud system. Therefore a combination of the tools we reviewed above is needed unless these capabilities can be captured within a single unified model.

2.3.1.3 Simulation-enabled MBSE

A visual/graphical M&S language, such as executable SysML, provides additional value beyond regular M&S tools which only provide limited visualizations of the system their simulating, For instance, the DoD has been using SysML system specifications for many value-added and visual-based artifacts for software-based systems such as: Authority to Operate (ATO), safety, airworthiness, and nuclear certifications to name a few. Furthermore, the Sandia report [60] concludes that there is a significant advantage to project performance when applying an MBSE approach compared to a paper-based SE approach. Therefore, simulation-enabled MBSE approaches using a visual-based language provides more value than M&S alone.

Many current MBSE methodologies use SysML as their main visual/graphical-based language. SysML [61] is valuable as a visual specification format that is semantically consistent compared to other drawing tools such as Powerpoint or Visio. However, SysML lacks the precise execution semantics for executable simulations. The Object Management Group (OMG) created a solution for this shortcoming by releasing the fUML standard [62]. Therefore, a simulation-enabled model using a visual language like SysML is possible when using a combination of SysML and fUML. The rest of this section will discuss different methods of accomplishing this.

Karban et al. [33], who developed ESEM for requirements verification, uses SysML modeling patterns that involve parametric diagrams for dynamic power roll-ups. However, in our experience with time-dependent MBSE-base M&S parametric diagrams do not capture the time-history of the simulation which makes it difficult to understand emergent system behaviors. These diagrams do not work for time-discretized simulations where events happen at a specific rate, rather than instantaneously like parametric diagrams in SysML. These diagrams can cause errors in time-dependent behavior, therefore these parametric roll-ups should only be used for instantaneous events.

An MBSE focused M&S for an air vehicle platform [63] similar to the legacy sensor fusion example demonstrates that executable SysML design models are capable of simulating data flow on a real-time MIL-STD-1553 (1553) aircraft data bus. The result of these system simulations informed the design decisions of the system engineers when adding functionality to a legacy system. The development effort successfully met the technical challenges of cost effectively building design models with sufficient fidelity to provide valid results. This reinforces our methodology of using SysML based M&S MBSE to design a complex data system.

2.3.2 System Analysis and Optimization

MDAO has been shown to accelerate the development of complex systems while using SysML based system model [64]. Additionally, OpenMDAO [65], an open source framework developed by NASA Glenn Research Center, has shown the advantages of executable SysML models integrated with a common data analytics environment using common APIs [66]. Additionally, several methodologies for architecture and system optimizations using SysML have been demonstrated on various systems such as submarine configurations [67] and satellite SoS architectures [68]. These demonstrations included the following four major steps. First, the requirements for the system must be defined. Second, a trade study that includes system constraints and an objective function to minimize is defined. Third, an architecture is defined that is not overly prescriptive and allows for various alternatives. Finally, the analyses of the objective function must by linked to the architecture model.

LaSorda et al. [68] demonstrated an MBSE approach using SysML and other computational analysis tools which optimized a real-world SoS architecture using multi-objective optimization. Individual systems were incorporated into an overall architecture model with appropriate quantified relationships to perform the SoS architecture optimization for parameters such as capability, schedule and cost. This approach by LaSorda et al. focuses on the pre-design phase of a system acquisition program where leverage on total program cost is the greatest. There methodology is in line with the goals of our research. However, with their application of multi-objective optimiza-

tions it is unclear what weights to apply to each objective function. Therefore, it is advantageous to provide many optimal solutions in terms of a Pareto-optimal curve, Pareto front, or a fuzzy Pareto frontier that allows us to identify dominant system characteristics [2]. These solutions provides a trade space for designers and stakeholders.

Herzig et al. [69] developed a model-based approach to identify Pareto-optimal solutions to explore the trade space of multi-spacecraft mission architectures. This automated SoS architecture optimization approach is centered around MBSE and was implemented using Ecore language for defining the meta-models. The following four objectives were used in the case study: minimize total mission cost, maximize coverage, minimize mission duration, and minimize length of non-empty transformations. Core to this approach is the automated synthesis of mission architecture candidates using model transformations, and the identification of non-optimal and Pareto-optimal solution candidates. The use of Pareto-optimal solutions is also in line with the goals of our research. However the application of their approach in a widely adopted visual language like SysML may improve the usability of the final architectural definition. Using SysML in MBSE can be beneficial to provide additional context, specifications, and requirements to the majority of system engineers that already understand SysML, as shown by Carroll and Malins [60].

At least two of largest CSPs have cloud optimization services available [70, 71], however these optimizations don't consider the on-prem or connectivity resources and costs involved with hybrid systems. These services must be part of a larger effort to optimize the entire system rather than just the cloud.

2.4 Data Collection, Analytics, & Algorithmics

The purpose of this section is to provide an overview of a subset of what has been pioneered in the past by experts in the field of data science and numerical methods for engineering. It is not the intention of this research to develop a novel approach to data mining, classification, machine learning, numerical methods, or systems-based analysis. Instead, we aim to highlight the strengths and weaknesses of what has already been developed and apply them appropriately to model emer-

gent data system behavior. Such behavior is complex to model and can appear to be stochastic. However, if we decompose the system into smaller, logical components with behaviors that are easier to model, then we can aggregate these components to estimate the complex system emergent behaviors. A recommendation from Meta-Algorithmics [72] is that "instead of finding the best possible intelligent algorithm, system, or engine for a task, the system architect should look for the best combination of algorithms, systems, or engines to provide the best accuracy, robustness, adaptability, cost, and so on."

2.4.1 Statistical Data Collection

In terms of collecting sufficient data to properly train ML models or collect a statistically significant amount of data samples to build a robust statistical models, He et al. [73] developed a statistics-based performance testing methodology for cloud applications. Their methodology, called PT4Cloud, results in a 95.4% accuracy on average while reducing the number of test runs by 62% when compared to extensive performance tests. This methodology employs non-parametric statistical approaches of likelihood theory and the bootstrap method to provide a reliable stopping condition to obtain highly accurate performance distributions with confidence bands. This testing methodology is a cost effective and reliable way to provide an empirical distribution of cloud-based performance data to create or train and validate models used to provide performance estimations.

2.4.2 Discretization, Parameterization, and Numerical Methods

Numerical methods provide engineers with a way to formulate mathematical problems so that they can be solved with arithmetic operations. One such example of this is in the field of Computational Fluid Dynamics (CFD) which includes one of mathematics hardest to solve set of equations known as the Navier-Stokes equations [74]. Within the field of CFD it is standard practice to solve the Navier-Stokes equations (a set of coupled, non-linear, partial differential equations) using a discretization numerical method [75]. This numerical method involves solving large numbers of tedious arithmetic calculations to solve the fluid flow around objects to estimate forces applied which is common in the aerospace industry. Simply put, one application of the the discretization

approach for CFD divides space into discrete finite-volumes that each have fluid input, output, storage or loss. Solving the fluid flow using this method can iteratively solve the Navier-Stokes equations and result in accurate estimations of forces and moments exerted on an object by the fluid flowing around it. This is computationally intensive and time consuming and it is common in MBSE to offload these numerical methods to professional CFD tools and import the results back into the MBSE tools afterwards.

The Quick Urban Industrial Complex (QUICURB) engineering tool and accompanying suite of tools is an example of applying discretization and parameterization to complex fluid flow to further simplify complex Navier-Stokes equations. QUICURB was developed to estimate fluid flow and plume dispersion within an urban environment [76–78]. Non-parameterized CFD solutions on the scale of a large urban area would normally take days to converge. QUICURB takes on the order of minutes within an acceptable accuracy for estimating plume dispersions within a city for emergency evacuation purposes. Parameterization is a mathematical process that expresses a system, process, or model in terms of parameters. For QUICURB, the upstream, leeward side, and rooftop air flow eddies around building are parameterized as a function of wind approach angle, speed, and air density for an initial condition, then the conservation of mass is solved. Furthermore, a method was developed to ingest large datasets to feed this parametrized model increased its accuracy [79, 80] via a Gaussian weighted-averaging technique. These examples show the decomposition of a complex problem into small, simple problems that are more easily modeled numerically. It also demonstrates that using numerical methods of discretization and parameterization complex behaviors can be solved using less computation power and time to do so. This example is also a reinforcement of the statement from Meta-Algorithmics [72] that we do need to find the best algorithm, instead look for the best combination of algorithms.

2.4.3 Machine Learning

This section reviews the research towards ML for time-series forecasting, sensor fusion, ML hyperparameter tuning, and ML Model Cards. ML and Deep Learning (DL) techniques can train

accurate time-series forecasting models for market forecast analysis applications or real-time sensor fusion.

The area of DL research has seen significant growth in popularity and application in recent years [81]. The data forecasting community has shown that DL-based approaches or a hybrid classical and DL approach can be more accurate compared to previous classical methods alone in time-series prediction algorithms [81]. N-BEATS [82], DeepAR [83], and Temporal Fusion Transformers for Interpretable Multi-Horizon Time Series Forecasting [84] are a few of the state-of-the-art approaches in forecasting models for accuracy, speed of training and speed of prediction with a trained model. However, these large, time-series forecasting models, such as DeepAR [83], take on the order of tens of minutes for a prediction with a pre-trained model. This latency can meet the needs of a design tool for predicting resource usage or cost for OLAP data system. However, this prediction latency would not meet the timing requirements for a real-time OLTP system.

For example, a -time sensor fusion algorithm on an embedded system has latency requirements on the order of milliseconds depending on the application. This prediction latency could also be detrimental to simulations of complex data systems which typically run 1,000's of times for Monte-Carlo analysis. Therefore, accuracy or length of time into the future for prediction may have to suffer to get the prediction latency down.

Section 2.4.3.1 focuses on different methods of tuning ML/DL hyperparameters while training ML/DL models. Section 2.4.3.2 reviews three different methods used for sensor fusion and time-series forecasting and highlights the need to share quantitative model performance and cost information with the sensor fusion industry for analysis on limited-resource systems. Many of the forecasting models in the sections below employ hyperparameter tuning and ensemble approach optimizations. However, it is unclear that any of the sensor fusion applications apply tuning or optimizations across multiple DL algorithms and data preprocessing approaches. Also, many of these forecasting models fail to track the DL resource usage for resource limited embedded systems typically found with sensor systems. Finally, Section 2.4.3.3 discusses the current methods for delivering ML model, data, and service information to the consumers of these products in a

responsible manner. This section notes the lack of standardization in the delivery of this information, lack of machine-readable formats being proposed, and lack of ML resources used during prediction/inference.

2.4.3.1 Hyperparameter Tuning

Hyperparameter tuning for ML algorithms such as Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs), or any other type is critical to ensure the best predictive performance is reached [85, 86]. However, the methodology described by Weerts et al. [85] requires on the order of 1,000 hyperparameter configurations trained per data set and ML algorithm which is a large computational cost. Additionally, Oreshkin et al. [82] emphasises the importance of optimizing the ensemble approaches to time-series forecasting. These best practices in time-series forecasting appear to be missing in many DL-based sensor fusion papers.

Two commonly used methods of hyperparameter tuning are grid search and random search. Grid search is common practice with three or fewer hyperparameters [87], whereas random search is used when there are more hyperparameters. Bergstra and Bengio [88] state that random search hyperparameter optimization is more efficient than grid search for solutions that are not sensitive to the hyperparameters being adjusted.

2.4.3.2 ML in Sensor Fusion

The two main DL techniques used in sensor fusion are CNNs and RNNs [87]. The first application of ML to sensor fusion applications reviewed here is for an autonomous driving vehicle. This system can be categorized as a resource-limited OLTP data system. Howard and Lewicki [89] utilized RNNs for sensor fusion in an autonomous driving vehicle, which increased certainty in the recognition, localization, and prediction of surrounding objects. This application used model performance to compare four different RNN-based approaches for sensor data fusion. However, this paper did not mention the resource cost of applying their algorithms.

Ensemble methods of layered RNNs and CNNs are increasing the performance and accuracy over basic ML algorithms. The DeepSense framework [90] uses layers of CNNs and RNNs to-

gether for sensor data fusion on embedded devices and also measure its performance and system costs. Yao et al. [90] noted an acceptable system trade-off of accuracy for energy on their battery-powered systems, where an acceptable degradation in accuracy consumed less energy. The results of this study provided latency and energy consumption for their inference model on two resource-limited devices, however it is not mentioned if these specifications are provided with the models in something similar to Model Cards. The absence of Model Cards makes it difficult for anyone to use or analyze the applicability of these pre-trained models on their system without any guidance. The latency of the DeepSense [90] CNNmodels were on the order of 100 milliseconds on a Nexus 5 mobile phone, while the random forest model took on the order of 300 milliseconds on an Intel Edison and about 50 milliseconds on the Nexus 5.

On the other end of the spectrum from the mobile capable DeepSense model, the DeepAR [83] models are large, ensemble, time-series forecasting algorithms and predictions take on the order of tens of minutes when running on a single cloud-based AWS *p2.xlarge* Elastic Compute Cloud (EC2) compute instance with 4 Central Processing Units (CPUs) and 1 Graphical Processing Unit (GPU). This specific RNN-based model is not used for sensor fusion, but if it is used on resource-limited devices in the future resource usage, cost, and latency should be tracked. Additional system analysis is needed as DL techniques become more advanced and the aerospace industry starts adopting them. This highlights that the application of DL models for sensor fusion systems are missing a standardized specification for model performance and cost in a machine-readable format to make it easier for automated system analysis techniques to read them.

Several papers demonstrate successful implementation of some form of DL algorithms for sensor fusion applications [83, 89, 90], but it was unclear if any method was used to find an optimized combination of ensemble layer configurations, data ingest approaches and hyperparameters combined.

2.4.3.3 Model Cards

This section compiles the previous work done toward the responsible use of AI/ML models in the commercial and government industries. There are many papers that comment on the need for

a standardized set of meta-data to ship with any pre-trained AI/ML model to engender trust. As stated in Richards et al. [91], recent work has outlined the need for increased transparency in AI for data sets [92–94], models [95], and services [91, 96]. Furthermore, Arnold et al. [96] envision this meta-data in *FactSheets* to contain purpose, performance, safety, security, and provenance information to be completed by AI service providers. Additionally, Richards et al. [91] describe a general methodology for creating FactSheets however, they fail to take the next step to provide a machine-readable format or schema to help automate ingest of the meta-data created by software applications.

Mitchell et al. [95] developed *Model Cards* as a step towards the responsible democratization of ML and related AI technology and increased transparency. They recommend a benchmark evaluation in a variety of conditions, such as different cultural, demographic, or phenotype groups that are relevant to the intended application domains. Mitchell et al. [95] discuss that Model Cards should be standardized and we intend to start this standardization process for RNN-based ML models.

Gebru et al. [93] developed *Datasheets for Datasets* that focus on the characteristics of ML data sets since these fundamentally influence the model's behavior. These Datasheets aim to provide data provenance for the ML community and increase the standardized processes for documenting ML data sets. The *Dataset Nutrition Label* [94] also focuses on data provenance to mitigate concerns of bias, safety, and security. Data provenance has significant safety impacts as it pertains to sensor fusion models used in aircraft. Mitchell et al. [95] also recommends using Datasheets as a complement to Model Cards which is also part of our research goal.

Blasch et al. [97] provides guidance for ranking/scoring AI/ML models for intended system evaluations in the Multisource AI Scorecard Table (MAST) for System Evaluation. However, like the other papers, it does not directly address ML costs for resource-limited systems.

Model Cards and Factsheets overlap and focus on purpose, performance, safety, security, transparency, and responsible use. For resource-limited systems these should be extended to include

system resource usage and costs to enable analysis of the ML model's impact on resource-limited systems or system simulation models.

Chapter 3

Yet Another MBSE Methodology (YAMM)

In this chapter we define the YAMM life cycle and its four stages originally shown in Figure 1.4 and reposted in Figure 3.1 in this section for convenience. YAMM refines and extends the Harmony aMBSE methodology [7] for a more prescriptive and practical application to software-centric data systems. The use case based design pattern in Harmony aMBSE provided the correct methodology foundation for the agile and DevOps principles common to most software-centric data systems. Use case based design produces a "narrow slice" of usable capability, or in agile software development terms, an MVP, which we define as the most basic version of a product with enough features to be usable by early customers, allowing developers to gather feedback and validate their product idea before investing heavily in further development. This MVP is then verified, validated, and deployed before moving onto the next MVP. In this manner, YAMM incrementally and iteratively adds MVP capabilities to the system. This design method is also known as a spiral life cycle development methodology as opposed to the waterfall methodology used by the SE V-model.

We apply this MVP concept to system level capabilities for each incremental iteration of YAMM. This MVP concept can also apply to a data system's hardware components within its Digital Infrastructure (DI), however the increments are slower than that of Software (SW) development increments.

One of YAMM's extensions to Harmony aMBSE is the adoption of the DevSecOps principles. DevSecOps is a collaboration between software developers, security, and system operators to automate continuous delivery of new software versions to an operational environment, while guaranteeing their correctness, reliability, and security [47]. We adapted these principles for YAMM by including the MBSE discipline into the collaboration and automation efforts to continuously deliver quality, dependable, and optimal data system MVP capabilities. Collaborating with the developer,security, and operation teams on the design pulls the system-specific knowledge curve to

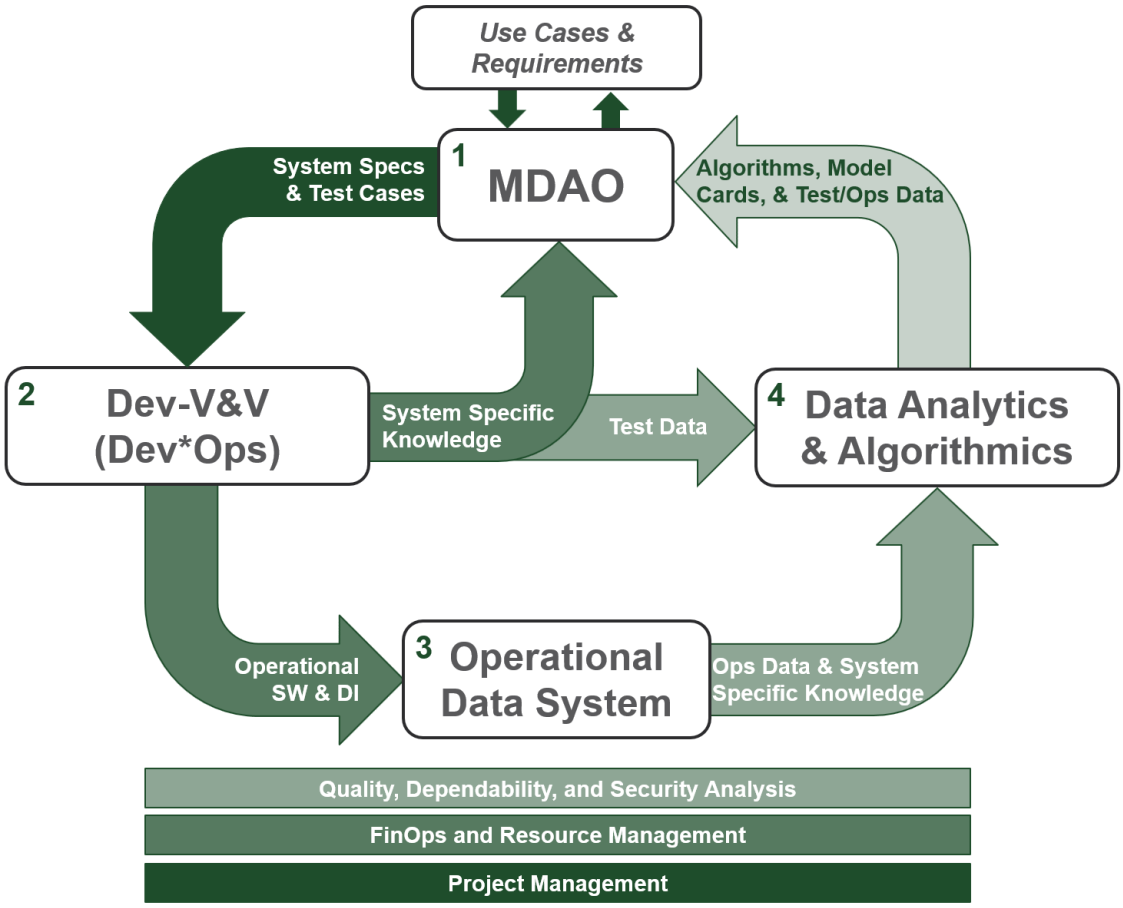


Figure 3.1: YAMM life cycle

the left while the ease of change is still high and costs incurred are low, as illustrated in Figure 1.1. This collaborative and multidisciplinary team feedback is shown in Figure 3.1 as the system specific knowledge flowing from stages two and three back to the SE team in stage one. The test and ops data collected from stages two and three are fed back through the analysis stage to provide empirically-base system specific knowledge back to the SE team. This empirical feedback increases system simulation accuracy and enables automated verification and validation of system simulations, which together improve system optimization effectiveness.

In addition to the four YAMM stages, we define the categories of data required for the application of YAMM in Tables 3.1 and 3.2. These tables include the definitions and examples of these data categories, the categories of data systems that would generate this data, and the YAMM stages where this data is collected or aggregated. These data and data system categories are referenced

throughout our description of YAMM, therefore these definitions and associations are valuable to understanding the data collected at each YAMM stage and passed to the next. For example, Table 3.1 shows that operational data is generated in both OLTP and HTAP type systems which are located in stage 3 of YAMM. Later, once that operational data is collected it aggregated in a OLAP system and the data is now considered analytical data which in YAMM stage 4 as shown in Table 3.2.

Table 3.1: Operational, OT, and DT Data, Data System Categories, Sources, and Uses

Data Category	**Data System Category**	**YAMM Source Stage #**	**Data Uses & Examples**
Operational Data	OLTP HTAP	-> 3 (Ops System) -> 3 (Ops System)	Real time transactional processes Point of sales **Examples:** Current inventory Real time sensor data
Test Data			
• **OT Data**	OLTP HTAP	-> 2 (Dev-V&V) -> 2 (Dev-V&V)	System validation of: • Real time transactional processes • Point of sales **Examples:** Current inventory Real time sensor test data
• **DT Data**	OLTP HTAP	-> 2 (Dev-V&V) -> 2 (Dev-V&V)	System verification of: • Real time transactional processes • Point of sales **Examples:** Current notional inventory Real time sensor test data Key performance parameters Key system attributes

The three data system categories, OLTP, OLAP, and the HTAP, listed in Tables 3.1 and 3.2 represent, respectively, operational transaction systems running in real time, analytical process systems that aggregate and analyze operational data, and a hybrid that combines these two categories to focus on real time or near real time analysis of operational data. While the focus of HTAP

Table 3.2: Analytical, Resource, and Metadata Data, Data System Categories, Sources, and Uses

Data Category	Data System Category	YAMM Source Stage #	Data Uses & Examples
Analytical Data	OLAP HTAP	-> 4 (Data Analytics) -> 2, 3 (V&V, Ops)	Operational & test data aggregation Business trend analysis Operations/Maintenance predictions Test analysis **Examples:** Sales or maintenance history (OT) Assess effectiveness & suitability (DT) Assess performance & function
Meta-analytical Data	OLAP	-> 2, 3 (V&V, Ops)	OLAP system usage data aggregation Analytical system trend analysis Analytical system resource predictions Analytical system test analysis **Examples:** ML training compute usage (OT) Assess effectiveness & suitability (DT) Assess performance & function
System Metadata	OLTP HTAP OLAP	-> 2, 3 -> 2, 3 -> 4	Information about system & data itself **Examples:** Data & recording formats Data sources & sizes
Resource Data	OLTP HTAP OLAP	-> 2, 3 -> 2, 3 -> 4	System resources consumed **Examples:** Compute, storage, network usage & costs

systems is to provide real time analysis of operational data, we do not foresee the need to analyze system metadata and resource data at a rate anywhere near real time. From our experiences during the development of YAMM, changes to a small data system enabled by the analysis of system metadata and resource data will typically happen on the order of months, but possibly weeks at the fastest. These delays are due to the humans in the loop that make the design, software, and test changes. However, with the proliferation of AI services, these delays may be shortened in the future.

In YAMM Process Flow 1, we define the high level YAMM process behavior that corresponds to the life cycle shown in Figure 3.1. This behavior is focused on new data systems in general, however this process can be easily adapted by legacy systems by starting midway through the

process. Legacy data systems with planned upgrades or optimizations would initially start the SE team working on step 1.3 while in parallel the Software Development (SW-Dev) and DI teams would start working to collect data in the operational data system in step 3.3.

YAMM Process Flow 1.

1 ***MDAO* Stage:**

1.1 Like many other MBSE methods the first step is the collection of system use cases and requirements by the SE team from the system stakeholders.

1.2 From stakeholder feedback, the SE team chooses the system architecture and models the system design at a high level of abstraction which contains the system structure, requirements, and use cases. This is sent to the multidisciplinary team in the ***Dev-V&V*** stage for initial collaboration, feedback, and sub-system design. This can also start the process of procuring long lead items, such as DI hardware, network approvals like an ATO or authorized connection, procurement contracts, staffing plans, etc. Feedback on requirements and use cases is provided back to the stakeholders continually.

1.3 The SE team designs or updates a system MVP in terms of structural and behavioral specifications, test cases, and data collection requirements. Once these are matured within the model they are passed to the SW-Dev and DI teams in the ***Dev-V&V*** stage.

1.4 While the rest of the multidisciplinary team is working on their initial downstream engineering designs the SE team iteratively develops and matures the UM simulation algorithms, functions, libraries, tests, and framework [25].

1.4.1 After the simulation runs and the tests are built updates made to the UM undergo V&V using the latest OT or operational data collected.

1.4.2 The SE team performs the system optimization and analysis and identify dominate system characteristics then decompose those into new specifications and incorporate them into the UM and deliver these updates to the ***Dev-V&V*** stage.

1.5 A System Readiness Assessment (SRA) is run on the current design which adds Technology Readiness Level (TRL) & Integration Readiness Level (IRL) metrics to model sub-systems. System Readiness Level (SRL) metrics are calculated and tracked to enable project management risk reduction for the system.

2 ***Dev-V&V* Stage:**

2.1 Once a system architecture or system specification MVP is delivered to the SW-Dev and DI teams they design the tests using DevSecOps principles, the data collection software, and the hardware, software, and services needed for the DI. Collaboration with the SE team continues throughout this stage.

2.1.1 In parallel, the DI team procures the hardware, software, and services needed for the system's DI. Long lead tasks, purchases, or approvals for software and the DI are started here.

2.2 After the DI team completes their designs they continuously develop, fix, and test the DI hardware, the DevSecOps and operational Software Defined Infrastructure (SDI), and the development, testing, and staging environments. These tests are automated using SDI DevSecOps Pipelines (DSOPs).

2.3 The DI team continuously deploys the SDI to the DevSecOps DI as updates are made.

2.4 The SW-Dev team continuously develops, fixes, and tests the operational software and data collection software in the DevSecOps DI environments using their operational software DSOP for automatic verification. They also provide feedback about the DevSecOps DI back to the DI team.

2.5 The DevSecOps DI resource data and system metadata are collected and delivered to the ***Data Analytics & Algorithmics*** stage.

2.6 System specific knowledge during software and DI design, development, and test is sent to the SE team in the ***MDAO*** stage which can start an MVP update process at step 1.3 if changes are required.

2.7 Any changes or updates to the SDI, operational software, or data collection software are continuously delivered to the operational system in the ***Operational Data System*** stage after being verified using the appropriate DSOP.

3 ***Operational Data System*** **Stage:**

3.1 The SW, SDI, and the DI are continuously deployed to the operational system as needed.

3.2 The end users begin to operate the system and provide feedback about specific MVPs or the system in general. If this feedback drives a design change, then it starts a new MVP process at step 1.3. If the feedback is a software defect, then it goes to step 2.4.

3.3 The operational resource data and system metadata is continually measured, collected, and delivered to the ***Data Analytics & Algorithmics*** stage.

4 ***Data Analytics & Algorithmics*** **Stage:**

4.1 Test and Ops data are analyzed to produce or update component algorithms and pass these to the ***MDAO*** stage along with processed operational data to V&V the UM. The fidelity of the algorithms created can increase with the maturity of the system.

5 **Continuous *YAMM Analysis & Optimization*:**

5.1 Continuous YAMM data collection, analysis, and optimization runs in parallel with each system MVP and throughout system's design, development, and sustainment stages to ensure optimal system costs and performance across all MVPs. This is to ensure an optimal system rather than a single optimal MVP. This represents maturing the simulation algorithms and UM accuracy in step 1.4 in parallel with the MVPs.

5.2 In addition to optimizing across MVPs this also captures system changes as user demand increases or use cases change. These new system trends will show up in the ***Data Analytics & Algorithmics*** stage which will start the YAMM life cycle again. New

component algorithms are created and a new process is started at step 1.3 with a new use case to model, optimize, and analyze.

5.3 In addition to new user trends and use cases, software can become more performant or defects fixed which can also lead to changes in system behavior which will also show up in the ***Data Analytics & Algorithmics*** stage. These performance updates may lead to updated algorithms which would be sent to step 1.4 to be incorporated into the model, then the system would be optimized and analyzed to see if any specifications will change.

5.4 These updates to system performance and behavior ensures the UM accurately reflects the data system's performance and costs continually. These small updates over the sustainment phase of a large system better enables future system analyses if new capabilities are added. In our experience with modernization programs for legacy systems it is easier to maintain software over the system's life than it is to fix software that hasn't been touched for years.

Depending on the size of the data system and organizational bandwidth, additional SW-Dev or DI teams can be provided multiple MVPs to work in parallel. Each new MVP starts the YAMM process at step 1.3. The time between YAMM MVP increments depends on the size of the data system, the size and skills of the SE, SW-Dev, and DI teams, and the maturity of the data system and data system technologies. For instance, these MVP time scales could be up to a year for mature and well optimized data systems, or on the order of weeks to months for a new data system being developed.

Figure 3.1 also illustrates YAMM's inheritance of quality assurance, dependability analysis, and project management from the Harmony aMBSE life cycle. Harmony aMBSE defines the dependability analysis as a combination of safety, reliability, and security analysis. Within YAMM we expand security analysis to collect artifacts towards obtaining a data system ATO. Within the DoD systems without an ATO are not approved to be used in an operational setting which is why we promoted security analysis as one of the foundational aspects of the YAMM life cycle. System

security must be designed into the system from the beginning and analyzed at every YAMM stage. Additionally, we added FinOps and resource management as foundational aspects to be designed and analyzed throughout the YAMM life cycle as illustrated in Figure 3.1.

Quality, dependable, and optimal YAMM SE designs are enabled through the automation of empirically-based system model verification and validation, system optimization techniques, system analyses that include Pareto analysis, trade space analysis, sensitivity analysis, and a SRA [9]. The SRA is a system technology and integration assessment that quantifies system robustness. We added this SRA to YAMM to extend the list of dependability analyses to include robustness. In Section 3.1 we define the first YAMM stage process which generates these quality, dependable, and optimal SE designs. The *MDAO* stage process itself contains three sub-stage processes that are iterative and automated themselves. The Unified Modeling approach we defined is the core sub-stage within the YAMM *MDAO* process that ties the system optimization and analysis together.

In Section 3.2, we describe the *Dev-V&V* stage which defines the test data collection process we have defined for a standard DevSecOps process. The analysis of test environment resource usage and cost data enables the early system optimizations in the *MDAO* stage and increases SE system specific knowledge early in the system life cycle. Collecting sufficient and quality data for the *Data Analytics & Algorithmics* stage is arguably more important that the algorithms themselves. Numerical model trustworthiness is proportional to the trust of the data it was built from.

Section 3.3 describes the *Operational Data System* stage, YAMM stage three, which focuses on the operational data collection process we have defined for the operational data system. This operational environment resource usage and cost data enables the increased accuracies of the empirically-based algorithm developed in the next YAMM stage and also validates the Unified Model created in the *MDAO* stage.

Section 3.4 describes the *Data Analytics & Algorithmics* stage, the final YAMM stage, that we defined to analyze the operational and test resource data and the system metadata to gain insights into the system, build algorithms that model system component behaviors, and define the Model Cards describing these algorithms and their resource usage. Each of the four YAMM life cycle

stages iterate independently of the other stages to incrementally deliver MVP capabilities in the manner of the DevSecOps philosophy.

3.1 *MDAO* Stage

The following sections provide a summary of the data flowing into and out of the YAMM *MDAO* stage. These sections also describe the details the UMA, system optimization, and system analysis sub-stages and their incremental and iterative cycles between them.

In this section we define our iterative and incremental MDAO technique that is the most crucial stage of the YAMM life cycle shown in Figure 3.1. Figure 3.2 is an illustration of the YAMM *MDAO* stage process and its three *MDAO* stages: Unified Modeling Approach, System Optimization, and System Analysis. This figure illustrates the ingestion of stakeholder use cases and requirements and the iterative process of providing feedback as the YAMM process is followed. This figure also shows the system specific knowledge that is fed into this stage from other YAMM stages in forms of collaboration feedback from the multidisciplinary teams and quantitative feedback from data collected in the *Dev-V&V* and *Data Analytics & Algorithmics* stages. Additionally, this figure shows the output of this stage in the context of system specifications and test cases that are fed to the multidisciplinary teams in the *Dev-V&V* stage.

The rest of this section defines each of these YAMM *MDAO* stages. Section 3.1.1 defines the UMA which is the process that iteratively and incrementally defines the use case based MVP design capabilities and the system simulations. In addition to stakeholder use cases and requirements the UMA receives empirically-base feedback from the YAMM *Data Analytics & Algorithmics* stage in the form of system component algorithms, parameters, Model Cards, test data, and operational data. This empirical system feedback is used to increase the accuracy of the UM through the incorporation of these empirically generated algorithms or parameters into the UM while the Model Cards are used to select which algorithm is uses the least amount of simulation compute resources in order to speed up optimization and analysis simulations. The operational and test data is used

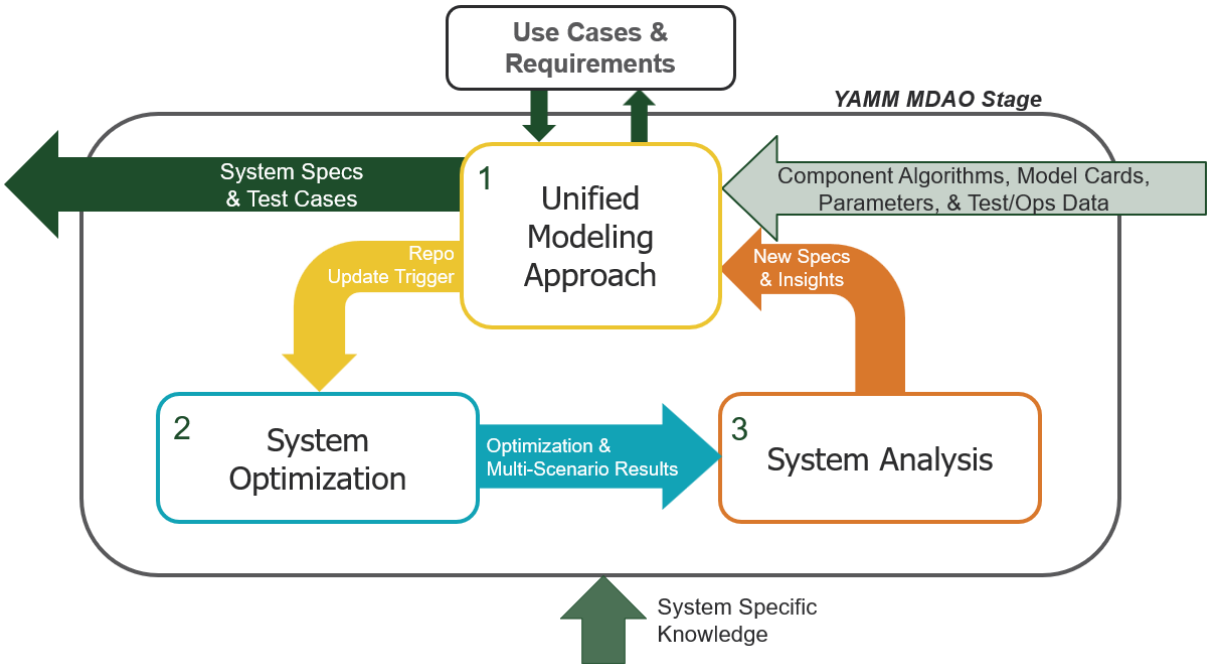


Figure 3.2: YAMM *MDAO* Stage

to validate the UM simulation through automated testing. Once tests are passed and the model is merged into its repository an update trigger notifies the System Optimization stage to begin.

Section 3.1.2 defines the System Optimization stage of the *MDAO* process. When automated, this stage begins running optimizations when it receives an update notification from the UM repository. This stage is also meant for manual exploratory optimization or running multiple scenarios for sensitivity analysis. The System Optimization stage passes these multi-scenario simulations and system optimization results to to the final *MDAO* stage for multiple analyses.

Section 3.1.3 defines the final *MDAO* stage which analyzes the optimization and multi-scenario results and passes new specifications and system insights back to the UMA stage. Once the new specifications and insights are built into the UM it is verified and validated and this *MDAO* process iterates again and until the system specifications and test cases are ready to pass to the YAMM *Dev-V&V* stage.

This entire *MDAO* process is repeated each time new use cases, requirements, algorithms, Model Cards, test/ops data, or system specific knowledge is delivered to this stage or, in terms of the larger YAMM Process Flow 1, a new MVP system design is scheduled.

3.1.1 Unified Modeling Approach

In the following sections we define the UMA, its focus and the assumptions made when we developed it (Section 3.1.1.1), the process and structure of the approach (Section 3.1.1.2), the method behind the decomposition of the data system, data system behaviors, simulation behaviors, and the algorithm functions (Section 3.1.1.3), the V&V techniques applied using a mixture of Test Driven Development (TDDev) and Test Driven Design (TDDesign) used for model development and system design (Section 3.1.1.4), and lastly we define the measurements used to quantify the UM simulation accuracy as compared to the test or ops data (Section 3.1.1.5).

3.1.1.1 Introduction

The value of MBSE as a method to simply visualize system engineering design has not been sufficiently proven [98], however system optimization been shown [60, 99] to be valuable. Additionally, M&S has been shown to increase the value of MBSE by verifying system specifications [63, 68, 100] early in the design life cycle. Furthermore, the combination of MOO and fuzzy Pareto analysis techniques for complex systems have been shown to provide additional value to MBSE, M&S, and system optimizations by generating Pareto-optimal solutions [69, 101–103] that also drive trade space analyses.

We have noted that many prominent MBSE trade study techniques are implemented using one specification model and at least one other simulation model usually accomplished with multiple tools requiring additional integration (e.g. Cameo, MATLAB, Model Center, and Excel, etc.) to perform architecture or system optimizations [68, 69, 103]. In these examples, the system simulation occurs outside of the core MBSE system specification model and tool leading to possible integration issues such as: interpretation errors, tool API version errors, or capability restrictions we've found when integrating multiple tools. Furthermore, the work to verify and validate these multi-tool integrated models increase as the number of tools increase. For these reasons we chose to combine the MBSE system specification model with the system simulation model(s) into a single unified model.

The most common problem with a single unified model is that most complex system design using MBSE have simulation tools built by specific engineering disciplines that are better suited for their specific disciplines than a simple MBSE modeling tool. Examples of this include CFD and structural analysis using Finite Element Analysis (FEA). These simulations are not meant for MBSE tools. However, as we have noted in Section 2.3.1 there are not sufficient tools for the design and simulation of ISs, cloud systems, or hybrid cloud systems. Therefore, we assume our UMA will be applied to data systems without the need for complex, discipline-specific, simulation and analysis. This approach reduces model and tool integration needs and only requires a common analysis environment with which to execute the simulations from.

We have noted that current modeling practices in MBSE lack the following five key modeling capabilities enabling iterative and incremental optimization and analysis techniques for complex software-centric data system within a single model:

1 Verification of functional specifications using simulation

2 Simulation validation approach for time-dependent system behavior

3 Modular, reusable, and non-proprietary modeling libraries

4 Ingestion of both scalar and time-series array input data

5 Compatible with common analysis environments and optimization techniques

Additionally, we gathered the top FEMMP evaluation criteria, that our UMA could affect, towards increasing the FEMMP evaluation score of YAMM and to make it more useful for the data system industry. This subset of FEMMP criteria, shown in Table 3.3, are extracted from the full list in Weilkiens [1]. Table 3.4 shows the combination of the five key modeling capabilities, the FEMMP criteria they meet, and the value added to YAMM by incorporating these capabilities.

In the next section we define our UMA which is developed to contain these five key modeling capabilities that also satisfy the FEMMP criteria. To enable the creation of a model with these key capabilities that also satisfies the FEMMP criteria we define a UMA process and meta-model structure. We derived this process and structure from software development practices and the

Table 3.3: Subset of FEMMP Evaluation Criteria

Title	Aspect	Description
Scalability	Methodology	How well does the model scale? (e.g. is it suitable for large projects?, can it "grow" with time without becoming cumbersome?)
Simulation	Methodology	How well does the methodology provide for integrated simulation?
Checking	Tool	Does the tool support consistent checking of the model? (e.g Automated detection of wrong content and/or formats, flagging of "loose ends"?)
Reusability	Tool	Does the tool allow reuse of any type of Modeling Element across projects? (e.g. sharing the same object with the same lifecycle in any project?)
Integration	Language	How well can the model be integrated with specialty engineering models?
MBSE-SE Info Exchange	Information	How well does the tool facilitate the exchange of the information between the MBSE domain and the non-MBSE tools
Meta-Model	Model	Does the methodology provide a (semantic) meta-model, ontology or similar?
Framework	Process	What views from the reference framework are used (MODAF, DODAF...)?

layered Service-Oriented Architecture (SOA) architecture. In the layered SOA architecture each layer provides functionality grouped by a common responsibility with explicit and loosely coupled interactions between the layers [104]. This architecture helps support a strong separation of duties that enable the flexibility and maintainability of a software system. These best practices make it easier to expand the size and complexity of software while reducing the possibility of breaking previous work.

3.1.1.2 UMA Process and Structure

The UMA is a simulation-enabled OOD approach. Figures 3.3 and 3.4 illustrate the UMA process and the UM metamodel structure, viewpoints, and content. The rest of this section will cover the UMA process illustrated in Figure 3.3 which shows its inputs from upstream YAMM stages and its outputs to the next YAMM and *MDAO* stages. This figure also illustrates the iterative UM generation in steps two through four performed in the modeling tool and testing stage of our TDDesign applied in step five from a common data analytics environment. Step five contains a set of unit and functional tests that verify the UM functions and utilities work correctly before

Table 3.4: Key Modeling Capabilities Mapped to FEMMP Criteria for Value to YAMM

Design Enabling Capabilities	FEMMP Criteria	Description of UMA Value Added to YAMM
1	Checking	A system model capable of verifying its functional specifications using simulation can increase the functional accuracy and quality of its system specifications [105, 106]
2	Simulation, Framework	A system model that accurately simulates time-dependent and software-based system behavior adds value for system engineers for generating derived requirements by identifying emergent system behaviors [107, 108] A clear distinction between specification and simulation viewpoints enhances the model users comprehension of the UM
3	Scalability, Reusability, Meta-Model	A system model with a clearly defined meta-model architecture using a modular, reusable, and non-proprietary set of modeling libraries enables future scalability in the size and complexity of the model which can save SE time and money [104, 109]
4	Scalability, Simulation, Reusability	A system model that accepts scalar data as input early in the design process provides system engineers useful insight into the emergent system behaviors [25]. A system model that also accepts empirical time-series inputs can better estimate real-world, time-dependent conditions. Early and continuous empirical feedback from the system can increase the rate of system specific knowledge early and throughout the system's life cycle [5]
5	Integration, MBSE-SE Info Exchange	A unified model compatible with common analysis environments and optimization techniques can enhance system performance, efficiency, and reduce costs [110]

allowing the model to be merged into the model repository. Once test or ops data is ingested this step also validates the model against this real world test/ops data and provides measurements of accuracy before merging the model. After the model has been merged into the main branch of the model repository a trigger is sent to the system optimization stage for system optimization and analysis while the system specifications and test cases are released to downstream engineers to start system development activities.

More specifically, we will define our novel UMA step by step and discuss the tight coupling of the process with the meta-model structure. Our novel, iterative, and incremental UMA process

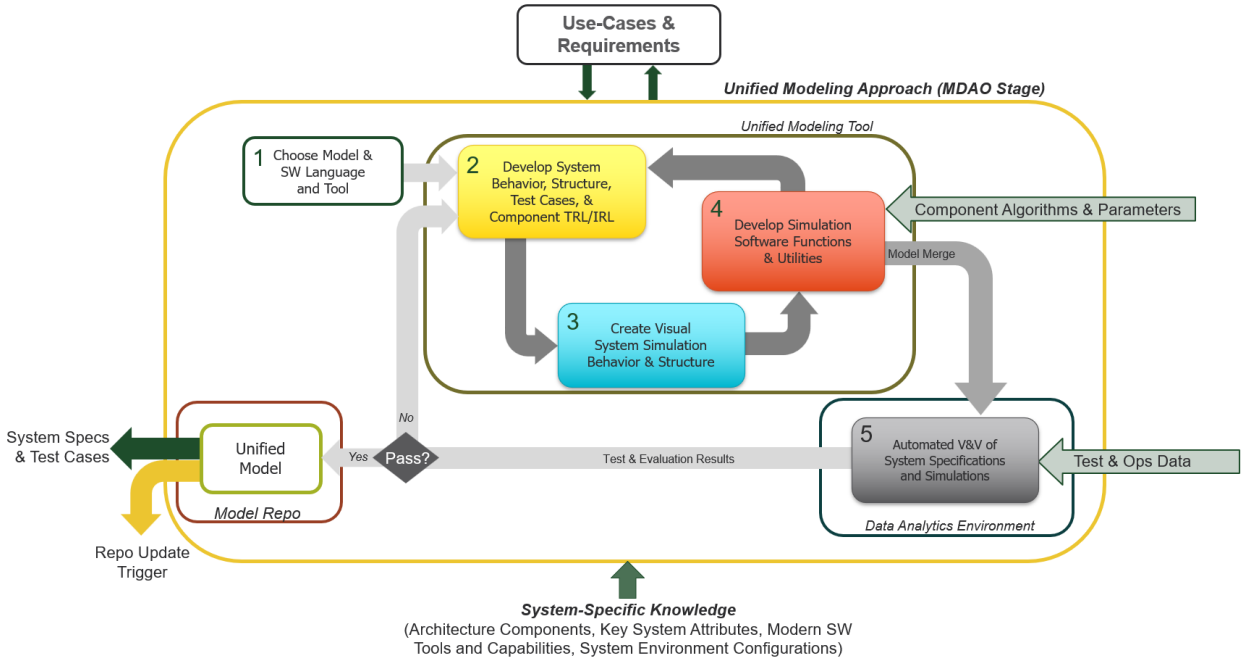


Figure 3.3: UMA Process (Stage One of the *MDAO* Stage)

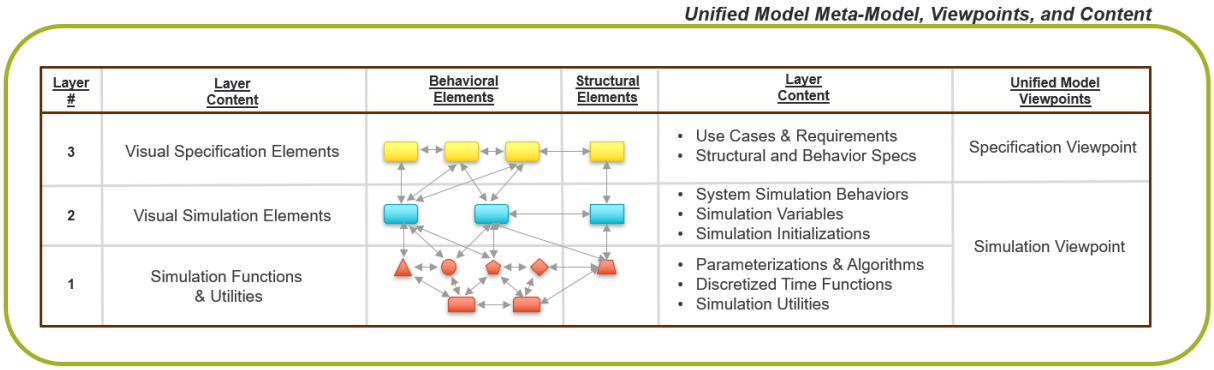


Figure 3.4: UMA Metamodel Structure, Viewpoints, and Content

starts at step one where the SE team must decide on the modeling language and tool as well as the SW language that will provide the most capabilities for the system specification, simulation, optimizations, and analyses for the system to be developed. These choices should also be flexible, cost effective, and user friendly for the organization and team. We summarize the approach we took to choose our languages and tools in the UMA application to the hybrid cloud case study in Chapter 5.

Once the languages and modeling tool are chosen, steps two through four are applied within the modeling tool as shown as the boundary around these steps in Figure 3.3. The UMA process from steps two through four is a method of top-down meets bottom-up model and software development that happens within the modeling tool chosen. This is represented by the color coding of process step two in Figure 3.3 with layer three in the UMA meta-model structure shown in Figure 3.4. The system architecture must be chosen and a rough outline defined first in step two and as MVPs are developed small slices of the UM are developed from the system architecture down through steps three and four, which correspond to the similarly colored layers two and one, respectively. Defining the system simulation behavior and structure in step three is dependent the previous step, however without the simulation functions and utilities developed in step four the simulation won't function. Therefore, these bottom layer functions and utilities are created to enable the simulation from the bottom-up. Since the development of the top and bottom layers seem to have to be done in parallel the UMA is an iterative process of refinement from steps two through four.

We apply TDDesign to this top-down meets bottom-up development by writing the unit and functional tests for the UMA simulation functions and utilities and tests that validate the entire UM, tool, and language against real world data. This happens in the larger iterative loop shown in Figure 3.3.

The UM structure as shown in Figure 3.4 defines three logical meta-model layers. The first and lowest novel UM meta-model layer we defined in Figure 3.4 is composed of libraries of functions and utilities using a common programming language that deliver advanced simulation services. One novel aspect of our UMA is the development and application of layer one that performs mathematical operations too complex for typical MBSE models. This layer contains functions of advanced empirically-based algorithms and parameterizations that enable flexibility of updating the simulation behavior rapidly by swapping out these algorithms for more accurate one. This is faster than using manual methods with visual languages like SysML or UML. These advanced functions are an aggregation of many modular smaller behaviors representing individual system components to ensure they are easily upgradable. We discuss this decomposition and aggregation of functions,

behaviors, and components in more detail in 3.1.1.3. In addition to being easily upgradable, the aggregation of many small algorithms with limited scope tend to be more accurate than one large, complex algorithm built for an entire system which we discuss further in 3.4. This layer also contains the utilities that create the discretized time series values in the structural elements that return value arrays that contain the state of the system as a function of time rather that scalar values at the end like many MBSE simulations. The value arrays enable post-simulation analysis and the calculation of objective function values for system optimization. These utilities also enable the size of the simulation time step to be adjusted as a model input from the analytics environment, which allows the SE team to adjust the fidelity of the simulation. Examples of structural elements and value arrays in layer one include the state history of storage size, storage cost, compute cost, and numbers of VMs consumed during processing. One example of a behavioral function in layer 1 is the empirically derived relationship that calculates the number of VMs needed to process a data file given a specific file format, size, software version, hardware, and resource constraints.

UM meta-model layers two and three in Figure 3.4 are composed of a visual-based Domain Specific Language (DSL) defining the system specifications and high-level simulation logic. These layers are found in most MBSE models and are appropriate to define with visual modeling languages like SysML and UML. As shown in the figure, layer two simulation behavioral elements are dependent on the modular behavioral algorithms and simulation functions in layer one as well as the value arrays within the simulation structural elements. The elements in this layer are closely tied to defining the system behavior specifications found in layer three. Layer three composed of the system specifications such as the requirements, use cases, system activities, parametrics, and system architectural structure. Elements in UM meta-model layer 3 define the structure and behavior of the system, however these elements are dependent on layers one and two to accurately simulate this system behavior within the system's defined structure.

In Figure 3.4 we define two viewpoints that separate the simulation elements from the specification elements for model users. This helps to provide a common understanding of the distinction between the system specification and simulation perspectives. The specification viewpoint

contains the requirements, logical & physical system architecture, structure, and the behavioral elements that define the system. This viewpoint is found in most MBSE models and can be split into many other viewpoints in other frameworks like Department of Defense Architecture Framework (DODAF) or Unified Architecture Framework (UAF). Another novel aspect to our approach is the definition of the simulation viewpoint which includes the simulation configurations, analysis blocks, simulation activities, opaque behaviors, and simulation functions and utilities. This allows insight into the simulation capabilities of the UM. Having the simulation and specifications in one ASOT makes it easier to keep track of simulation and specification versions since they are saved in one model.

3.1.1.3 System and Simulation Decomposition

In this section we describe our UMA strategy of decomposition and simulation aggregation for use cases, component behaviors, simulation behaviors, and the simulation function and associated algorithms. Figure 3.5 illustrates the decomposition of system use cases into sub-system use cases then into component behaviors and finally into components within the UM. The yellow squares in this figure represent structural specification elements, yellow ovals represent use cases, and yellow squares with rounded corners represent specification behavior or activities, whereas the red squares represent structural simulation elements that begin simulation execution for the yellow element it is associated with. Additionally, this figure shows the allocations of one-to-many system use cases to the data system, one-to-many sub-system use cases to each data sub-system, and one-to-many component behaviors to to each data system component. Each system use case, sub-system use case and component behavior is color coded and located in the system specification UMA layer three in Figure 3.4.

Figure 3.6 shows the internal composition and simulation of a use case from Figure 3.5. These use cases include one-to-many component behaviors allocated to at least one component. Each component behavior includes one-to-many simulation behaviors which in turn calls simulation functions and utilities during execution. The order of simulation in Figure 3.6 is labeled from one to nine and executes each time the simulation analysis block in this figure is started by an analysis

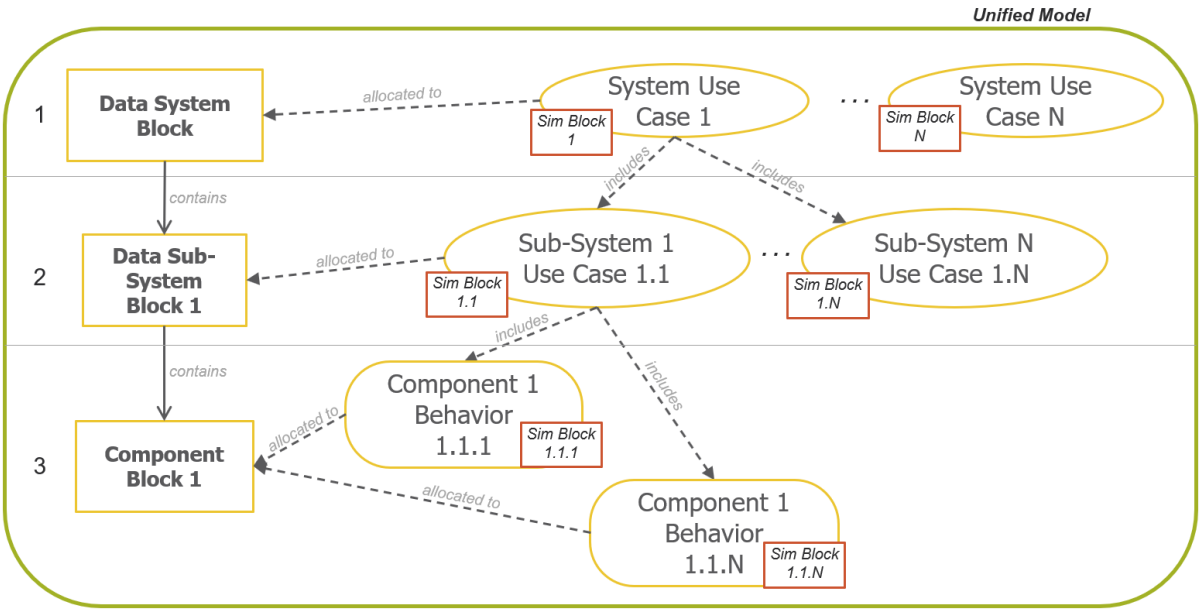


Figure 3.5: UMA Use Case and Behavior Decomposition

from the data analytics environment. 3.5 shows the decomposition of each use case from the top system use cases. We have built it in this way to execute any level of simulation that we want to analyze. This means that we could execute "System Use Case 1" and it would in turn execute its own simulation as well as all the sub-system use cases and component simulation that it includes. We did this to aggregate many modular simulations and to also break each behavior into small pieces that are easier to accurately model than the whole system at once. Aggregating these small accurate behaviors in this way increases our chances of accurately capturing complex emergent behavior [25].

Hardware and software components are modeled as structural components, however most of the time behaviors come from the software running on the hardware. In Figure 3.6 each use case simulation is started by the system and simulation inputs provided from the analytics environment to the UM in step one. One of the simulation inputs dictate which use case to simulate and that one is executed in steps two and three respectively. Each component behavior within the executing use case and each simulation behavior within each of those component behaviors will execute in the order they are defined in the model which is represented in this example by steps four and five,

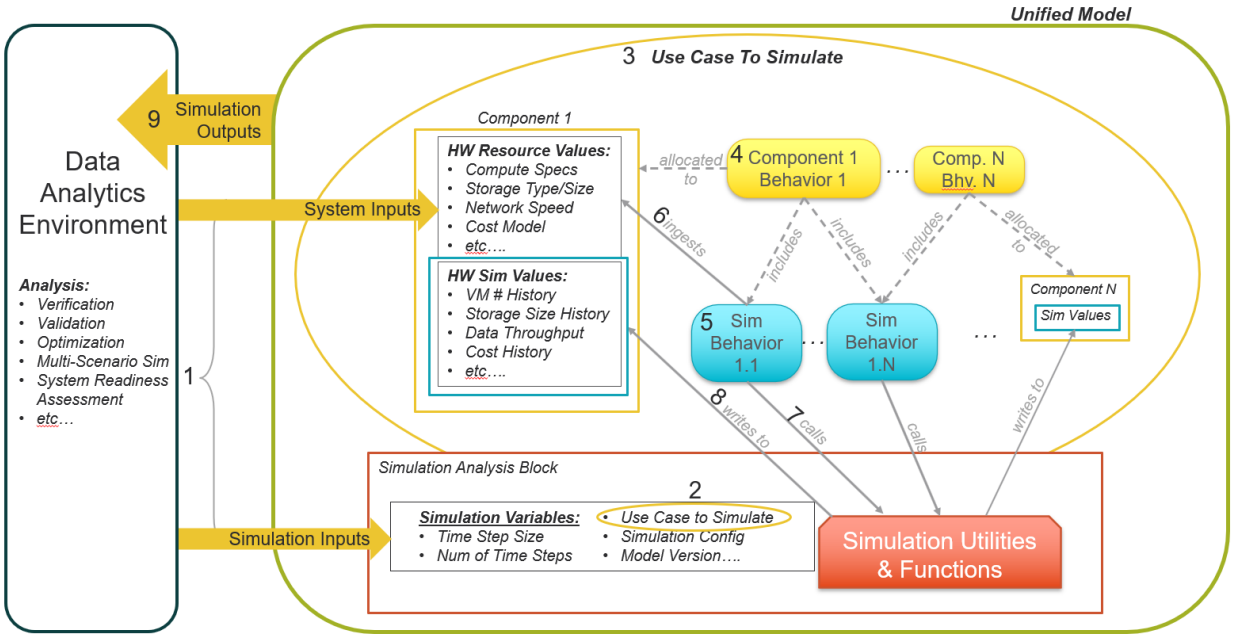


Figure 3.6: UMA Use Case and Component Simulation

respectively. We do not show execution of the rest of the component and simulation behaviors in steps to keep this figure clean, but in a model they would also execute.

We enable modularity in component and simulation behavior through the allocation of these behaviors to structural components. This allows each behavior to execute in the context of this structural component and use its values as inputs and write to its simulation history value arrays. Figure 3.6 shows that "Component 1 Behavior 1" is allocated to the structural "Component 1" which includes its own resource specification values. This allocation provides an execution context which is passed to its simulation behaviors during execution. These simulation behaviors execute in this same context enables them to ingest the correct resource values in step six. During execution this simulation behavior passes these variables and execution context to the simulation functions and utilities in step seven. These functions and utilities execute their algorithms and write the results back to the execution context simulation values in step eight.

This simulation happens for each discrete time step and stores the state of these components as a function of time in the simulation values. At each time step the simulation values for each child component are summed and rolled-up to the parent use case where those are summed and rolled

up to the highest level executing use case using parametric functions to provide system level totals such as cost, storage size, compute, or network resources. Once the simulation is complete the simulation values of all components, sub-systems, and systems are returned in step nine back to the analytics environment and analysis algorithm which originally started the simulation execution.

3.1.1.4 UMA V&V

In this section we discuss our application of a mixture of TDDev andTDDesign for model development and system design, respectively. We discuss how we ensure the specifications are semantically verified through the use of the model tool and our set of core specification and simulation abstract objects, how we apply TDDev to our UMA for the verification of the simulation functions and utilities, and how we use TDDesign when designing system test cases that collect data which validates the UM and tool against real world test or operational data. These V&V and TDDev are separate from the applications employed in YAMM *Dev-V&V* stage which we will discuss in Section 3.2.

We start the discussion of system specification V&V with our UMA bottom-up approach that begins with our UM abstract resource and simulation model library which defines the key structural objects, functions, and utilities. Below, we define abstract objects, their inheritances, and how we leverage common MBSE tool consistency checking to verify the correct object types are used properly Figure 3.7 shows the UM abstract objects where all squares represent structural elements while the yellow, teal, and red colors correspond to the UMA metamodel layers three through one, respectively, as shown in Figure 3.4. Our UM resource and simulation model library is used in each data system model developed and, as with other software libraries, continually updated with new MVPs as needed.

Through multiple object inheritance, we can create most data system resource specifications (software or hardware) by combining the UMA Abstract Compute, Storage, Cost, Cloud Cost, and Throughput Objects. For example, an on-prem server would be an inheritance of the abstract Compute, Storage, and Cost objects. However, if this server was decomposed into separate storage, archive storage, and high-performance compute server subsystems each of those subsystems would

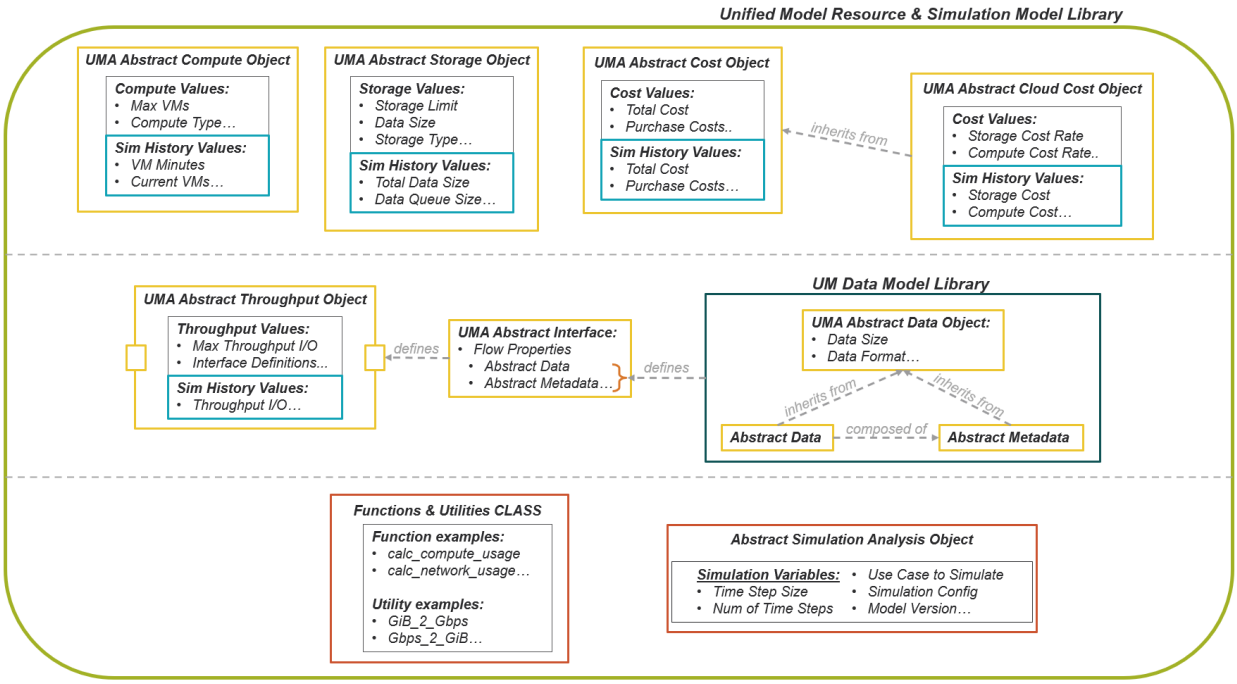


Figure 3.7: UMA Resource and Simulation Model Library Concept

now need to inherit the abstract throughput object to enable data flow and communications between the three subsystems. Furthermore, if this server workload moved to the cloud, then it would inherit from the abstract cloud cost object instead of the abstract cost object. Figure 3.7 shows that the abstract cloud cost object is a specialization of (i.e. inherits from) the abstract cost object then adds cloud cost specification values such as compute and storage cost rates as a couple examples.

Furthermore, Figure 3.7 illustrates the relationship between the abstract throughput object, the abstract interface, and the UM data model library. The data model library is yet another UM library that defines system data types and expands with need and is a dependency of the UM resource and simulation model library. For the UMA we define this model library's abstract data and abstract metadata such that all data types after this inherit from these two data types. We then define the flow properties of the abstract interface as either one of these two data types and define the abstract throughput object I/O ports with this abstract interface, as shown in the middle horizontal layer of Figure 3.7. The

The last model library contained in the UMA is shown on the bottom horizontal layer of Figure 3.7. The functions and utilities written in a common software language are contained in a class (or function depending on the language) which is also a dependency of the UM resource and simulation model library. Finally, the abstract simulation analysis object in this figure is the generalization of each simulation analysis block shown in Figures 3.5 and 3.6.

These object definitions and inheritances are central to ensure UM type consistency verification. We define these objects so that most quality MBSE modeling tools can identify type consistency errors if an object has no inheritance, or the wrong inheritance, from these abstract objects. For example, if a new data object does not inherit from either the abstract data or metadata object, but it is a data flow between two throughput objects, as illustrated with the green arrow in Figure 3.8, then the tool type consistency verification will flag it as a warning or error. This is part of the definition that only abstract data or metadata objects (or inherited objects) can flow between abstract throughput objects as defined by the interface definition flow property as shown in Figure 3.7. We extend this consistency checking philosophy to all aspects of the UM abstract object definitions and continually upgrade our libraries as needed.

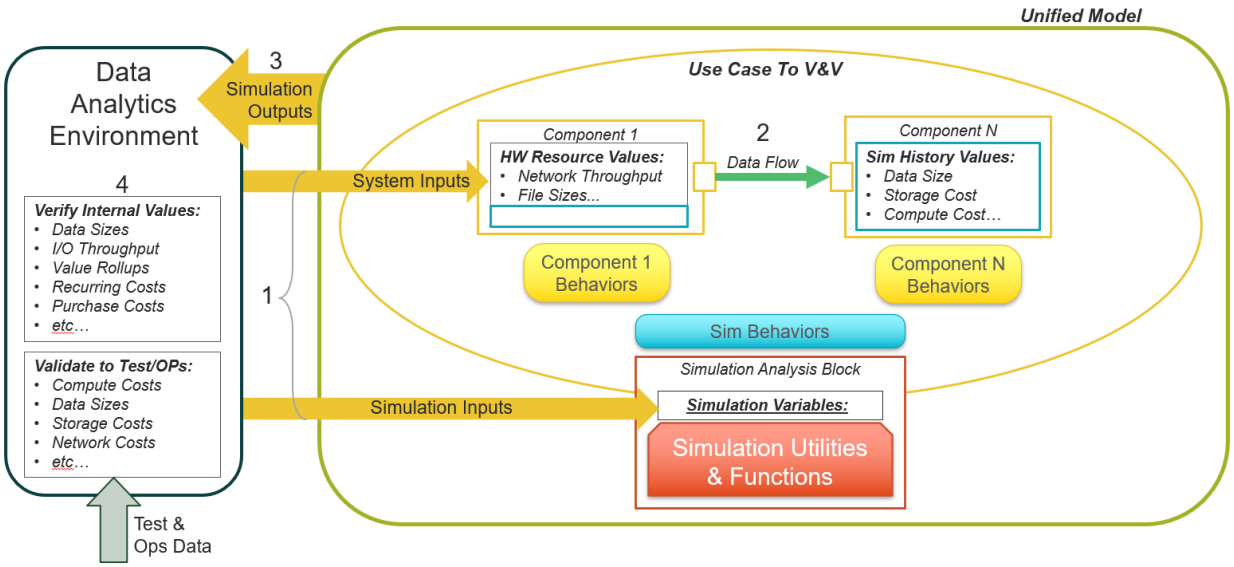


Figure 3.8: UMA V&V technique towards TDDesign

We now move onto functional specification and simulation verification which is not covered by a tool's consistency checks. Figure 3.8 illustrates the method we deploy for the UMA to use simulation to improve functional specification verification of the model. This figure illustrates the UM simulation from an analytics environment in the same manner as Figure 3.6, however in Figure 3.8 we execute tests with known simuation outputs for the inputs provided. In this figure we show an example of data flow between two components with their associated component behaviors, simulation behaviors, and simulation functions and utilities. The functional verification tests written in the analytics environment simulate this example use case and verify the following example values: 1) component 1 throughput out matches component 2 throughput in, 2) data transferred from one component to the other are equal in size and timing, 3) all value rollups from component to system level are correct, 4) data queue sizes get reduced by the same size of data that is being process and removed from the queue, and 5) recurring and purchase costs occur at the right intervals.

In addition to the correct simulation outputs being verified, this test technique verifies if specific components have inherited the correct values from the abstract objects. Errors in the simulation occur values are missing from the specification object (execution context). This can happen if the specification object did not inherit values from the correct abstract object. For example, calculating cost on an object that did not inherit from the abstract cloud cost object will not have the "Storage Cost Rate" available to calculate the storage cost.

These functional verification tests are executed each time the UM gets updated as shown in step five in 3.3. The number of verification tests grow as capabilities are added to the model.

We have focused on TDDev and UM verification up to this point in this section and now move on to TDDesign and UM validation. We refer back to the illustration of the UMA in Figure 3.3 where the data system test cases are developed alongside the system specifications and these test cases are sent to the YAMM *Dev-V&V* stage as shown in Figure 3.1. The SW-Dev and DI teams develop these test cases, collect test and operational data from them, and this data makes it back to

the YAMM *MDAO* stage and, more specifically, makes it back to the UMA analytics environment as shown in Figure 3.8.

We validate the combination of the UM specifications, simulations, and tool by running a simulation that mimics the test conditions of each test case and comparing it to the test and/or ops data collected. We quantify the model error by calculating the accuracy of the data system's costs (compute, storage, network, purchase, etc.) and its resource usage (compute, storage size, network throughput, internet throughput, etc.) as an aggregate and as a function of time. In Section 3.1.1.5 we define the methods of quantifying the error of the UM with respect to the test or ops data.

3.1.1.5 UMA Quantitative Accuracy Measurements

In this section we discuss the two types of accuracy measurements we use for the UMA's time dependent models. The first type is an aggregate or total error. This represents the error of a total cumulative value (e.g. cost, storage, compute, etc.) over the simulation duration as compared to the same test or ops data. For example, this can be used to report the simulation accuracy in terms of total cost over a year. For this type of aggregate accuracy measurement we use total percent error as defined in (3.1), where Δy is the residual ($|y_{real} - y_{sim}|$ between the real y_{real} and simulation y_{sim} data., For percent error, the residual is calculated using the final cumulative quantities of the real data and sim data, respectively.

$$P_{error} = 100 \, \frac{\Delta y}{y_{real}} \tag{3.1}$$

The second type of accuracy measurement represents how well the simulation captured time dependent behaviors rather than cumulative values. This calculates the simulation residual at each time step then aggregates those residuals over the total duration. For the UMA we define two different accuracy measurements that each have their strengths.

The first time dependent accuracy measurement is the RMSE which represents the average residual over the time frame of the simulation. We use RMSE as define by (3.2), where n is the number of data points, or time steps, and Δy_t is the residual at time t. We chose RMSE instead of

Mean Squared Error (MSE) so that accuracy measurements can be represented in the same units as the values being compared. This provides more context to the reader than MSE which is in $units^2$. However, MSE is widely used in the fields of statistics and machine learning so users should choose based on what is standard in their field.

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(\Delta y_t)^2} \tag{3.2}$$

The second time dependent accuracy measurement we define for the UMA is the residual entropy, $H(\Delta y)$. In information theory [111], entropy is a measure of the randomness of a signal, so a completely random signal contains maximum entropy, H_{max}. We calculate simulation entropy to quantify the amount of non-random residuals in the simulation, or, simply put, when all we have left between real and simulation is randomness, then we have simulated the actual information [72]. We use the definition in (3.3) to calculate the residual entropy, where the probability of residuals, $p(\Delta y)$, is calculated on a histogram of N bins which defines max entropy as $H_{max} = \ln(N)$.

$$H(\Delta y) = -\sum_{i=1}^{N} p_i(\Delta y)\ \ln p_i(\Delta y) \tag{3.3}$$

We also calculate the percent max entropy error using the simulation residual entropy $H(\Delta y)$ compared to the max entropy and calculated using (3.1). For the UMA we use the entropy, max entropy, and percent max entropy error to provide a quantitative measure of the non-random behaviors the simulations captured.

In summary, we define the UMA to report simulation accuracy using the following quantitative measurements for each resource usage and cost value simulated: total percent error, RMSE, entropy, max entropy, and percent max entropy error.

3.1.2 System Optimization

System optimization can generate an optimal design solution for data systems which can save organizations millions of dollars per year. However, without accurate estimations of the variables

driving system costs these optimal design solutions will also not be accurate and could result is wasted investments. For this reason we reemphasize the importance of the previous UMA stage and its focus of continually improving the accuracy of its solutions through empirical feedback. Equally important is the iterative and incremental nature of the entire YAMM *MDAO* (Figure 3.2) stage to continually analyze the effects of the empirical feedback on the optimal solutions. For instance, if an MVP increased the speed and reduced the resource usage of a large software capability in the cloud it would change the cost, performance, and behavior of that system component. Capturing this new cost, performance, and behavior in the model accurately, then running system optimization could result in a new optimal system solution that drives other software changes in the system. This is one example that drives the need to automate this System Optimization stage and the System Analysis stage while we continually improve system performance and reduce costs as the system is being developed.

The process of system optimization includes four major steps:

1 Define the requirements for the system

2 Define a system that is not overly prescriptive and allows for various alternatives

3 Define a trade study which includes system constraints and an objective function to minimize

4 Link the objective function analysis to the architecture or system model

This technique has been successfully demonstrated using MBSE and SysML on various systems and SoS [67,68]. Within YAMM, steps 1 and 2 are completed in the UMA stage, step 3 is defined in this System Optimization stage, and step 4 is accomplished in the System Analysis stage discussed in Section 3.1.3.

Most complex data systems have multiple and opposing objective functions which require a multi-objective optimization technique that produces a range of Pareto-Optimal solutions [112]. One example of opposing objective functions is the relationship between performance and cost in the cloud. Using archival storage decreases read access speeds which reduce performance, however

archival storage is usually an order of magnitude cheaper. Therefore multi-objective optimization Pareto-Solutions wouldn't provide a single answer to optimize both cost and performance, rather it provides the trade space to allow decision makers to weight the importance of cost vs performance for them. Additionally Pareto-Frontiers may show an apex, or "knee-in-the-curve" where the best balance of these objectives is found. Our technique of multi-objective optimization using the UM is illustrated in Figure 3.9 which shows the model update trigger and system specific knowledge as inputs and the optimization results as outputs.

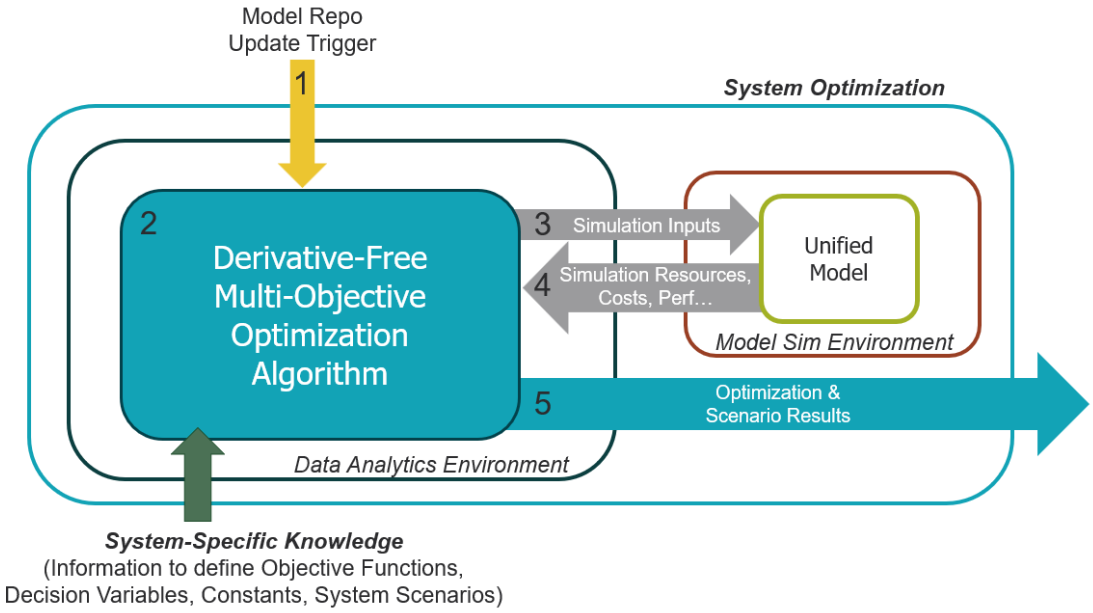


Figure 3.9: YAMM System MOO Process Enabled by the Unified Model

The rest of this section describes the system optimization technique we defined for YAMM, where Section 3.1.2.1 describes the approach we take to define the trade studies, system constraints, and objective functions as laid out above by step 3 of the system optimization process. Figure 3.9 shows the optimization and scenario results from this process getting passed to the next stage, System Analysis, which analyzes this data and links it back to the UM and finishes step 4 of the system optimization process listed above.

In Section 3.1.2.2 we outline several optimization algorithms compatible with our UMA, list the strengths and weaknesses of each, then define the one that best fits our MDAO strategy and, by extension, YAMM.

Finally, Section 3.1.2.3 we discuss the types of optimization results generate with our System Optimization process which are passed to the System Analysis stage. We describe the results which include Pareto-Optimal solutions, Fuzzy Pareto Frontiers, and Multi-Scenario simulations and also provide an overview of their uses in analysis.

3.1.2.1 Trade Study and Objective Functions

We discuss our method of defining objective functions, decision variables, constraints, and trade studies for the YAMM *Dev-V&V* stage in this section. Furthermore, frequent input of system specific knowledge from the multidisciplinary team, as shown in Figure 3.9, increases the SE team's ability to create well defined objective functions, constraints, and system scenarios. It also increase their ability to identify the correct decision variables and constants for the system optimization process.

Objective functions are the functions that you want to either minimize or maximize for your system. For example, our main focus of discussion has been to maximize system performance and minimize cost. However each SE team needs to collect stakeholder input and consider their data system carefully to identify other measurable objective functions. These typically include objectives like security, quality, dependability, and data value. Generalized examples of cost and performance objective functions given in (3.4) and (3.5), respectively, show how complex and tightly coupled these function are and why simulation is needed to be able to calculate them. We define $storage_c$, $compute_c$, and $network_c$ as the storage, compute, and network costs of the system, where each are define as functions of time t, data size s resource type r, number of Create, Read, Update, Delete (CRUD) actions a, software sw, or data format dw. Similarly, we define $storage_p$, $compute_p$, and $network_p$ as the storage, compute, and network performance of the system.

$$cost = \sum (storage_c(t, s, r, a, sw),\ compute_c(t, s, r, sw, df),\ network_c(t, s, r, sw)...) \quad (3.4)$$

$$perf = \sum (storage_p(t, s, r, a),\ compute_p(t, s, r, sw, df),\ network_p(t, s, r, sw)...) \quad (3.5)$$

We define each trade study by a combination of use cases as defined in Section 3.1.1.3 and a set of decision variables which are controllable parameters within the data system that are adjustable. The effects from adjusting these decision variables on the system objective functions are the focus of the trade study analysis. We enable easy and modular trade study definitions in our UMA by pairing the simulation analysis blocks with use cases which contain the component's specifications and behaviors to fully execute a simulation as shown in Figure 3.6. To start the trade study the SE team chooses the decision variables and define their range. From the analytics environment the SE team can loop through all simulation permutations generated by the combination of decision variables with values in their ranges. In this figure, each decision variable permutation is provided as system inputs to each component in the use case, step 1, and executed until simulation results are returned in step 9.

An example of this would be choosing network speed and storage type as decision variables each with three discrete choices, [1Gbps, 5Gbps, and 10Gbps] and [Hot, Cold, Archive], respectively. This would result in 9 permutations of simulation results which equate to 9 different objective function values of cost and performance. In this case the constraints provide a set of discrete choices, rather than a continuous range with max and min values. Each simulation will most likely have a combination of continuous and discrete decision variables and a set of constants to got with them. The constants in this example would be the hardware or monthly costs to enable the network speeds along with the read/write speeds, latency, and the cost rates of the different storage types.

Additional constraints within the data system are built into the specifications and simulations by definition. In Figure 3.6 these constraints are enabled by the definition of the component resource values, component behaviors, simulation behavior, and the logic built into the simulation functions.

The SE team is responsible not being too prescriptive and ensuring they are enabling trade studies while defining the model in the UMA. We defined the UMA to generate models specifically that would be modular and flexible which in the context of trade studies and optimizations ensure the SE team has the ability to generate a model capable of running many different analyses of their data system.

3.1.2.2 Derivative-Free Optimization Algorithms

A multi-objective optimization algorithm provides a set of Pareto-optimal solutions from a system simulation since an exhaustive search or brute force method is prohibitively expensive in compute time and resources. The simulation solutions are not well defined mathematical functions and therefore derivative-based optimization functions do not work on these types of problems. Additionally, many input variables into these data systems are discrete values rather than continuous values, such as internet speed, number of servers, number of nodes/server, number of nodes/cluster, etc., making this a mixed integer problem. Therefore the use of discrete optimization algorithms are required.

We review several derivative-free and discrete optimization algorithms compatible with our UMA technique. These are all capable of producing Pareto-optimal or Fuzzy Pareto Frontier solutions, however they have their own strengths and weakness. Derivative-free algorithms are the only optimizations that are compatible with discrete simulations that we generate with the UMA. We run the discrete simulations since there are not mathematical functions that can be written to describe these complex data systems, therefore without mathematical functions there are no derivatives.

The Compass search algorithm is a local search algorithm that is mathematically defined and is deterministic. It is parallelizable which can speed up the process when using large, complex, and discrete simulations. However, convergence of Compass search is slower than a linear rate of convergence.

The Dividing Rectangles (DIRECT) algorithm is a global search technique that is mathematically defined and is also deterministic. However, this algorithm is often prohibitively expensive

with its exhaustive search technique and its cost that exponentially increases with the number of design variables. Most data system simulations will require many design variables, therefore this method does not scale well for our situation.

The Genetic algorithm uses a global search technique defined by a set of heuristics with stochastic properties. The convergence of this type of algorithm is slow, with the best members often changing so slowly that multiple "generations" pass by with little to no change in objective values.

The Particle Swarm Optimization (PSO) algorithm is similar to Genetic algorithms and is also a global search technique defined by a set of heuristics with stochastic properties. However, convergence for PSOs is typically faster than Genetic algorithms.

Out of the optimization algorithms above, we choose PSO for our optimization technique for its faster convergence rates and global search capability.

3.1.2.3 Format of Optimization Results

Pareto-optimal solutions provide insights into the system's trade space needed for system engineers to make informed design decisions. Figure 3.10 shows an example trade space from a Guidance, Navigation, and Control (GNC) system from Cameron [2]. This figure shows mass versus reliability for more than 20,000 architectures analyzed and the Pareto frontier highlighted with a solid line. As defined by Cameron [2], the key concept for the Pareto front is dominance: The set of non-dominated architectures is called the Pareto frontier. An architecture A1 strongly dominates another A2 if A1 is better than A2 in all metrics. If A1 is at least as good as A2 in all metrics, and better than A2 in at least one metric, we say that A1 weakly dominates A2. If no architecture dominates an architecture A1, then A1 is said to be non-dominated.

In many cases, the Pareto-optimal solutions, or Pareto front, also provides the colloquial "knee in the curve", or apex, that represents a balanced tradeoff of both objective functions in a trade-study. In the example in Figure 3.10 two apex occur above 8 "9's in reliability". The first apex is approximately 8.4 while the second is around 9.7.

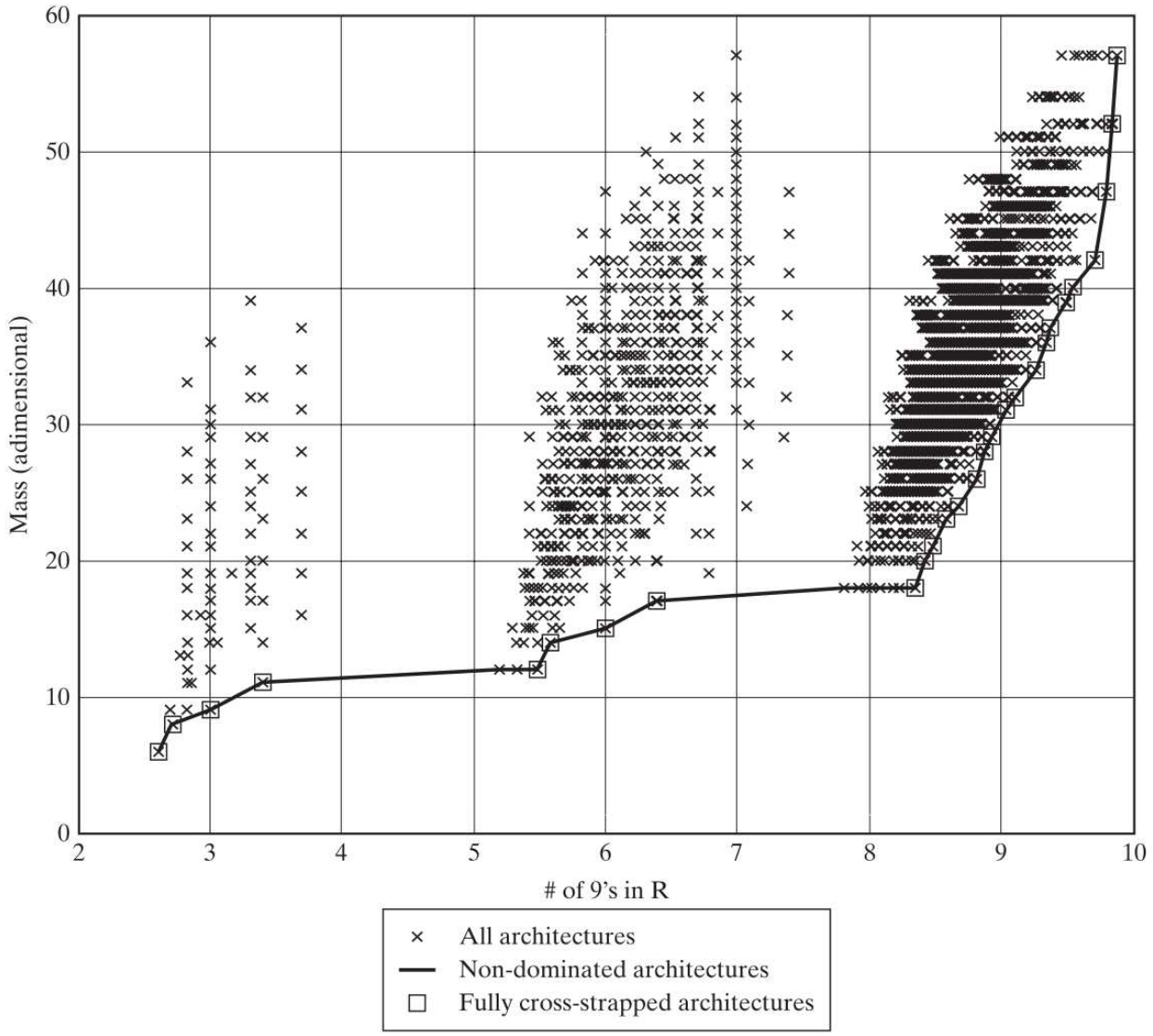


Figure 3.10: Pareto Frontier Example [2]

Dominated architectures or systems may not be a poor choice since the strongly dominated architectures/systems are sometimes the least robust in responding to variations in model assumptions [113]. Therefore the concept of the fuzzy Pareto frontier was introduced to capture the dominated architectures/systems near the Pareto front. The fuzzy Pareto frontier is the first cut at reducing the trade space.

The goal of the fuzzy Pareto frontier is to extract the architecture or system characteristics that are common for non-dominated architectures, these are referred to as dominate system character-

istics. In Section 3.1.3.1 we describe two trade space analysis methods employed using this fuzzy Pareto frontier.

We define the requirements in this stage to run multiple simulation scenarios. Running multiple scenarios in addition to the fuzzy Pareto frontier results are required for the sensitivity analysis that is further described in Section 3.1.3.2. Each scenario is defined as running the same simulations only with different modeling assumptions. For example, we refer back to the GNC example [2] and generate two different scenarios to compare. The first scenario is run with an assumption that there no penalty for dissimilar components, whereas the second scenario is run with a dissimilar component penalty added to architectures with dissimilar components connected. Figure 3.9 step five shows the output of the fuzzy Pareto frontier optimization results as well as the multi-scenario results to the next *MDAO* stage that we describe in Section 3.1.3.

3.1.3 System Analysis

This section describes our System Analysis techniques employed on the Pareto optimization (Section 3.1.3.1) and multi-scenario (3.1.3.2) results generated and described in Section 3.1.2.3. Figure 3.11 illustrates these analysis techniques in steps one and two. A third analysis technique shown at step three in this figure extracts component TRL and IRL data from the system model (UM) to assess the technology readiness and robustness of the system through the SRA process (3.1.3.3). Steps one through three are another incremental process that iterates until it is determined that dominant system characteristics or configurations are robust enough to include in the UM as newly derived system specifications.

3.1.3.1 Pareto & Trade Study Analysis

As mentioned in Section 3.1.2.3 there are two methods of analysis to extract dominant system characteristics from Pareto optimization results for step one in Figure 3.11. We describe both of these methods in this section and the trade study analysis. One method of capturing dominant system characteristic is to plot the systems that contain common characteristic until you find the ones that lie on the Pareto front the most. In Figure 3.10 this was done on this GNC example for

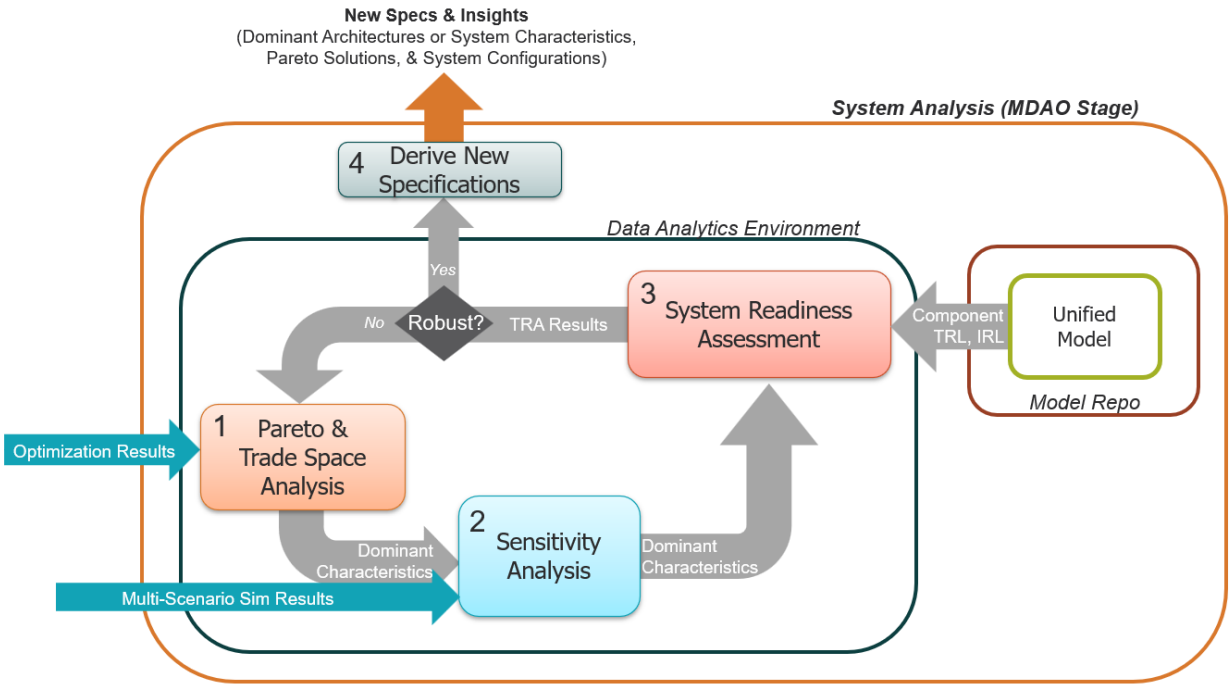


Figure 3.11: YAMM MDAO System Analysis Process

an architecture characteristic known as cross-strapping. This figure shows that all Pareto-optimal solutions are fully cross-strapped architectures by the squares plotted which means the full cross-strapping is a dominant system characteristic for this example.

Quantitative methods are a second method of capturing dominate system characteristics and provide a way for system engineers to eliminate poor performing architectures and highlight ones that have these characteristics. Table 3.5 shows a table quantifying dominant system characteristics.

Table 3.5: Percentage of Architecture from the GNC Example [2] on Pareto Front

GNC Architecture Feature	**% architectures with Feature in trade space**	**% architectures of Pareto front with this Feature**	**% architectures with this Feature on Pareto front**
Fully Cross Strapped	2%	100%	7%
Homogeneous	7%	33%	1%
Uses 1 or more Sensor C	59%	56%	0.1%

Table 3.5 contains the percentages of architectures on the GNC Pareto front [2]. This table confirms that the fully cross-strapped feature is a dominant system characteristic. Only 2% of the architectures in the trade space are fully cross-strapped, however all non-dominated architectures are fully cross-strapped. Furthermore, the homogeneous feature is only present on 7% of trade space architectures and is present in 33% of the Pareto front, which isn't a necessary condition to lie on this front, but it is a disproportionally high. This is suggesting it is a candidate for a dominate system characteristic when looking at the fuzzy Pareto frontier.

Conversely, 59% of the architectures in the trade space use one or more sensor type C, but about 56% of non-dominated architectures have this feature. This clearly indicates that this feature is not a dominate system characteristic.

We adopt the technique of calculating the fuzzy Pareto frontier by using a relaxing Pareto filter condition to include the slightly dominated architectures which are very near the Pareto front. This technique is defined by Cameron [2] and computes the true Pareto front, then discards all points that have a Euclidean distance greater than the filter threshold. For instance, any point greater than 10% Euclidean distance in a fully normalized objective space.

Trade space analysis from optimization results also happen in step one in Figure 3.11. A common feature in trade spaces is the presence of solution clusters [2]. These clusters represent a collection of architectures with similar features that affect their objective functions and leave large gaps between other clusters and architectures. Figure 3.12 shows an example of trade space clusters using the same GNC example as shown in Figure 3.10. These clusters are driven by an emergent property of the architecture.

These clusters identify a family of architectures with similar decisions and can lead to cluster-based, or set-based, s-Pareto frontiers [114]. These s-Pareto frontier can be used to improve architecture or system selection in design. The s-Pareto frontier-based concept selection method can be characterized as one that capitalizes on the benefits of computational optimization during the conceptual design phase before a general design has been chosen [114]. These clusters can be identified through human identification or through a variety of clustering algorithms.

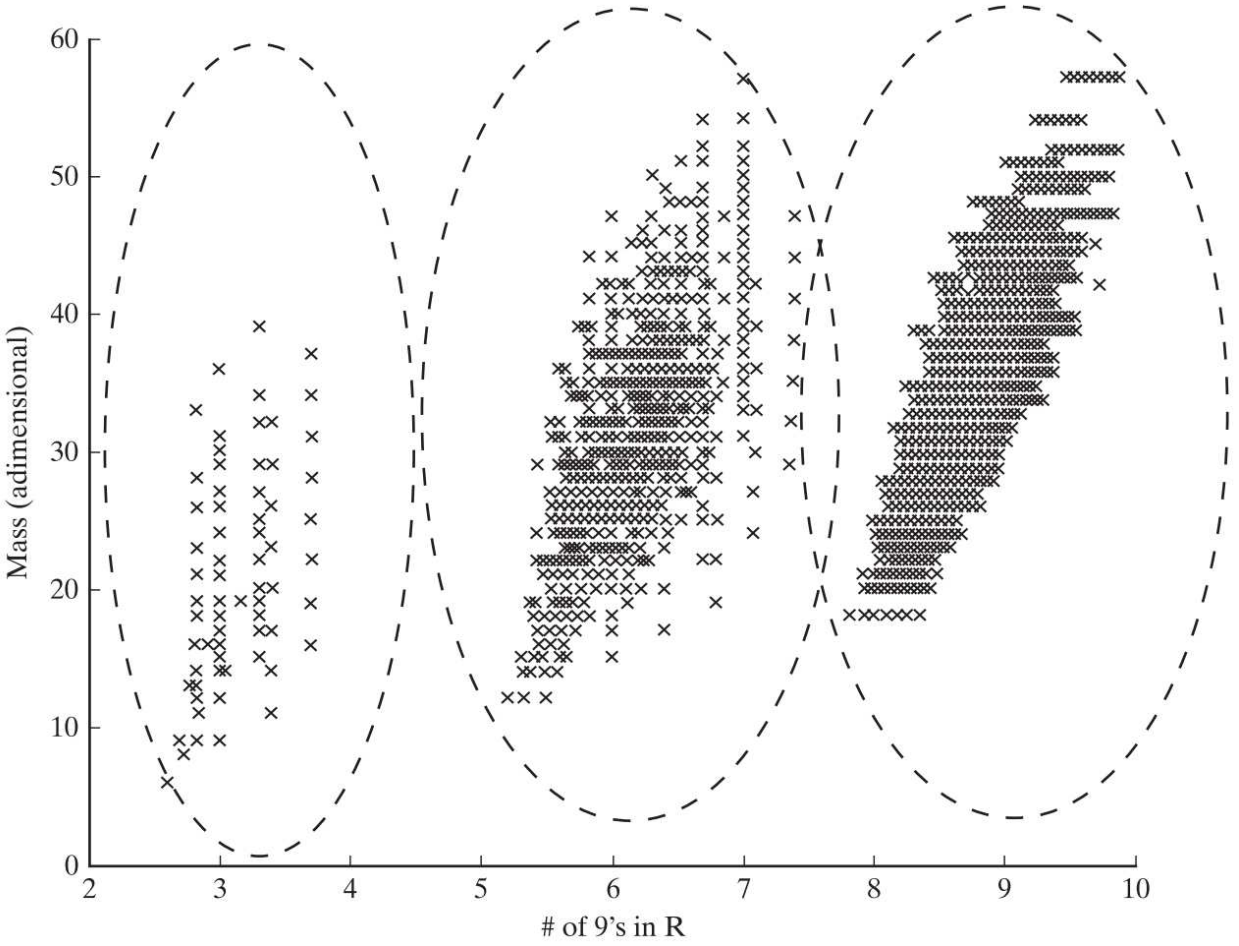


Figure 3.12: GNC Example of Trade Space Clusters [2]

Therefore, the need to analyze the fuzzy Pareto or s-Pareto frontiers depends on the stage in which you are in for the data system design. Once an architectural design has been chosen, we define YAMM process to begin to analyze fuzzy Pareto frontiers for dominant system characteristics rather than choosing between architectural features and the same goes for s-Pareto frontiers.

3.1.3.2 Sensitivity Analysis

In this section we define the sensitivity analysis that is run on the scenario data that is fed into this System Analysis stage from the System Optimization stage. At the conclusion of the fuzzy Pareto frontier and trade study analyses, it is important to understand and quantify the sensitivity of the dominant system characteristics identified to different simulation and model assumptions. Each new simulation run with a different assumption is defined as a new scenario.

In Section 3.1.2.3 we described two different model assumptions, or scenarios, from the GNC example [2]. These two scenario results are shown in Figures 3.13 and 3.14 where the assumption that a penalty for dissimilar components is zero versus a penalty of nine for dissimilar components, respectively.

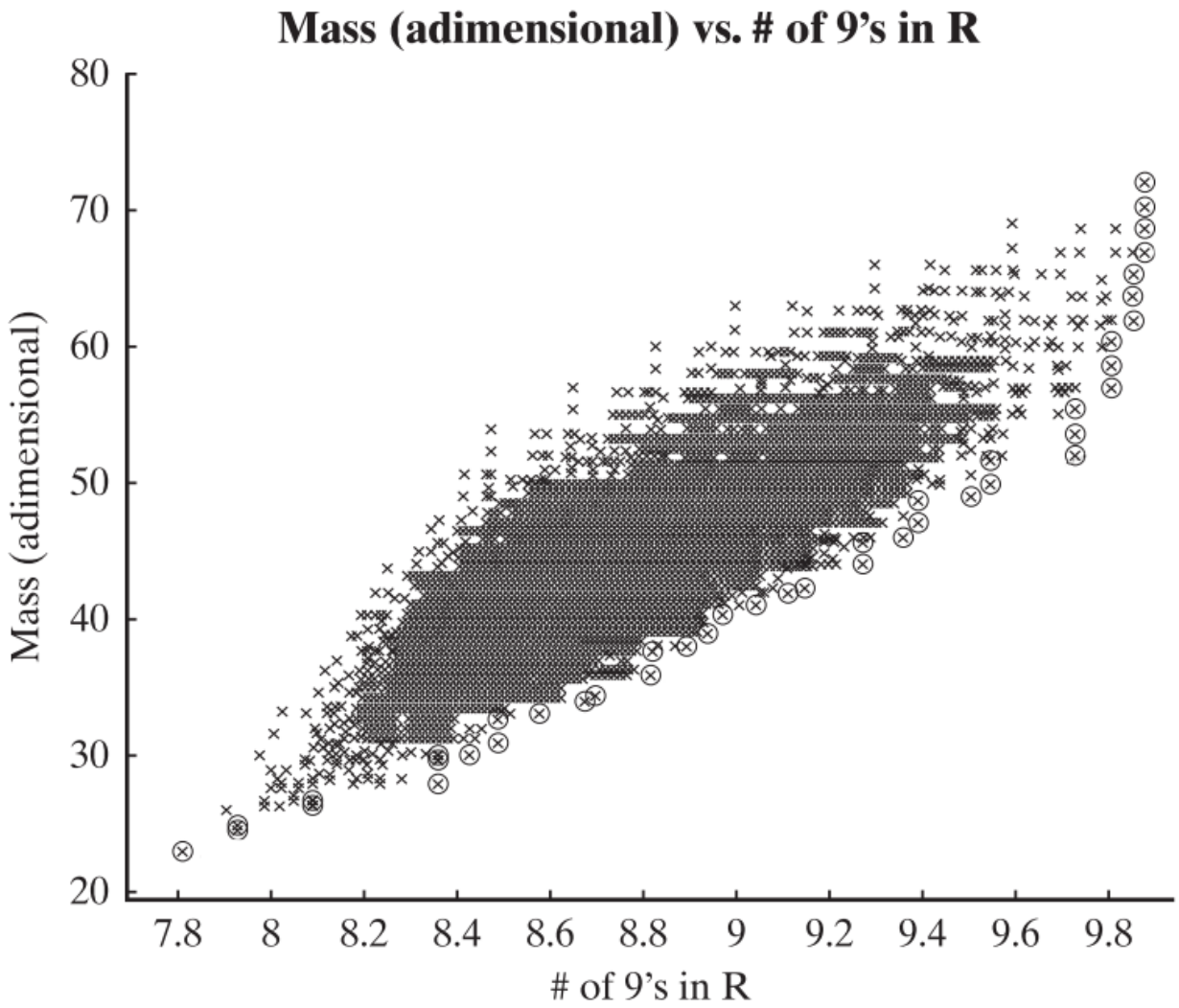


Figure 3.13: GNC Example of Scenario One with Dissimilar Component Penalty = 0 [2]

Figure 3.13 shows very little stratification for this solution cluster. Stratification are groups of points for which one of the metrics is constant while the other varies. Figure 3.14 with the dissimilar component penalty of nine produces stratification of the solutions. Additionally, it also

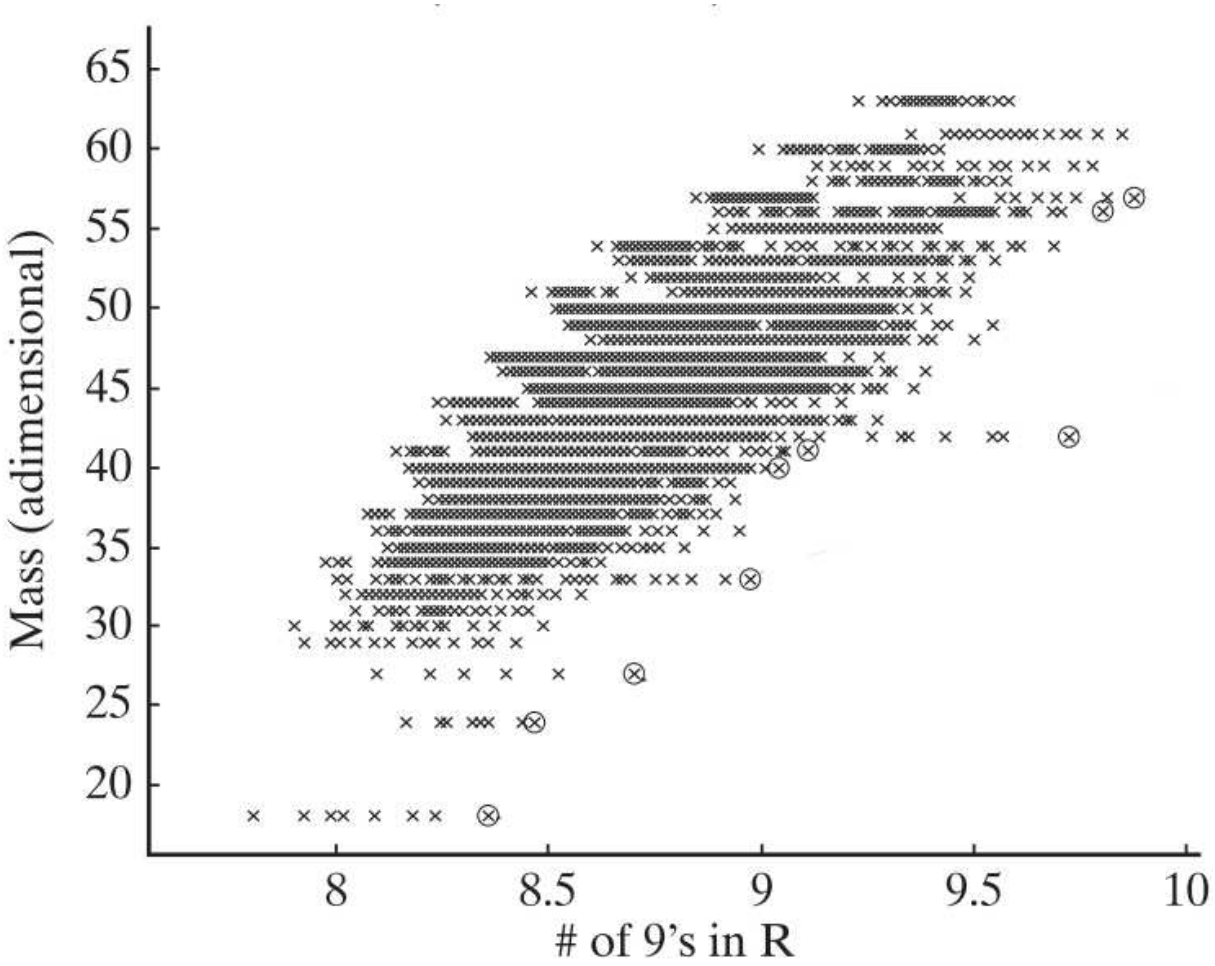


Figure 3.14: GNC Example of Scenario Two with Dissimilar Component Penalty = 9 [2]

shows several solutions that stick out to the right much more severely than solutions shown in Figure 3.13. These new solutions that stick out further to the right and generate a new Pareto front are the architectures with the homogeneous feature shown in Table 3.5.

This sensitivity analysis example demonstrates the importance of running separate scenarios and re-analyzing the dominate system characteristic results from the fuzzy Pareto and trade study analyses. It also demonstrates that the homogeneous feature that was not fully dominant became fully dominant when additional fidelity and new assumptions were added to the model. For these reasons we define this System Analysis stage to include a sensitivity analysis.

3.1.3.3 System Readiness Assessment

We describe an overview of the SRA engineering process and the ways in which it can help improve performance management for systems and aid decision makers in identifying programmatic and technical risk areas [9, 115, 116]. During the development of large, complex data systems it is critical to measure and track the system's readiness (SRL) along its life cycle to ensure the system and its associated technologies are maturing at the rate expected. Measuring and tracking the SRL enable more effective system development management and integration that can lead to shortened delivery timelines [9]. Additionally, empirical cross-referencing applied to the SRL framework can support more realistic cost and schedule forecasting early in a program development [116].

The SRL is calculated using a combination of the system's component-to-component IRLs and the system's component TRLs. The TRL is a systematic metric used by the DoD to assess the maturity of a particular technology and allow consistent comparisons of maturity between different technology types [117]. From the guide to Technology Readiness Assessment (TRA) best practices developed by NASA [8] Figure 3.15 provides a quick look at the TRA levels. Typically a TRL of 6 or higher is required for a technology to be integrated into a flight system, or in our case an operational data system.

The IRL is a metric that allows a risk assessment based on integration characteristics of the configuration item of interest [115,118,119] and was originally developed for as a management tool used to build computer networks [118]. The IRL definitions that relate to the TRL definitions are shown in 3.6. An IRL assessment is done for each connection between internal system components as well as the external facing components.

In addition to the TRA the SRA provides the following additional benefits: 1) Measures the readiness of all system components (considers all elements equally critical), 2) Focuses on the readiness of integration between components internal to the system and requires readiness understanding of other external dependencies, 3) Is performed at multiple times over the course of the system life cycle and not just at major milestones [9]. The IRLs and TRLs are assessed and assigned to component integrations and components within the UM as they relate and map to each

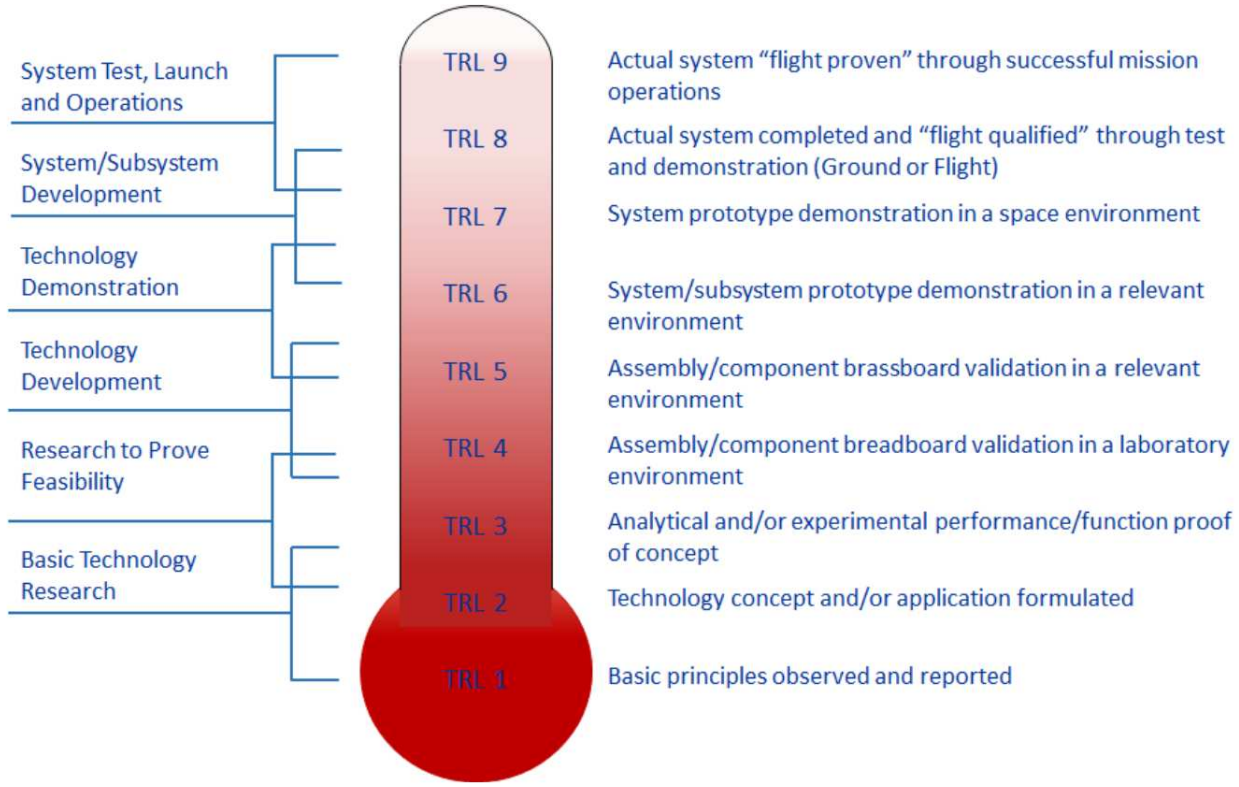


Figure 3.15: Thermometer Scale for NASA's TRLs [8]

other using the example shown in Figure 3.16. For example, all IRLs in this figure get mapped to a 10-by-10 matrix where the rows and columns are components 1 through 10. The IRL in between Components 1 & 7 is assigned to the $(1, 7)$ and $(7, 1)$ locations in this matrix.

After the component integrations IRL and components TRL are assigned in the UM, then the SRL is calculated in step three as illustrated in 3.11. The SRL matrix is calculated using (3.6), where the SRL matrix is the product of the IRL diagonally symmetric matrix and the TRL matrix is the vector of all n technology scores aligned with the IRL matrix.

$$[SRL]_{n\times 1} = [IRL]_{n\times n} \times [TRL]_{n\times 1} \tag{3.6}$$

Next, the component SRL is calculated using (3.7), where m_i is equal to the number of integrations for component i. For example, Component 8 in Figure 3.16 has 4 integrations, since the 'integration' with itself is counted, therefore $m_8 = 4$.

Table 3.6: TRL and IRL Definitions

TRL & IRL	TRL Definition	IRL Definition
9	Actual system proven through successful mission operations	Integration is Mission Proven through successful mission operations
8	Actual system completed and qualified through test and demonstration	Actual integration completed and Mission Qualified through test and demonstration in the system environment
7	System prototype demonstration in an operational environment	The integration of technologies has been Verified and Validated with sufficient detail to be actionable
6	System/subsystem model or prototype demonstration in a relevant environment	The integrating technologies can Accept, Translate, and Structure Information for its intended application
5	Component and/or breadboard validation in [a] relevant environment	There is sufficient Control between technologies necessary to establish, manage, and terminate the integration
4	Component and/or breadboard validation in [a] laboratory environment	There is sufficient detail in the Quality and Assurance of the integration between technologies
3	Analytical and experimental critical function and/or characteristic proof of concept	There is Compatibility (i.e., common language) between technologies to orderly and efficiently integrate and interact
2	Technology concept and/or application formulated	There is some level of specificity to characterize the Interaction (i.e., ability to influence) between technologies through their interface
1	Basic principles observed and reported	An Interface between technologies has been identified with sufficient detail to allow characterization of the relationship.

$$Component\ SRL_i = \frac{[SRL]_i}{m_i} \tag{3.7}$$

Finally, the composite SRL is calculated using (3.8), where n is equal to the number of components in the system. Using this equation the composite SRL will be a value between 0 and 1, therefore it is converted to a scale of 1-9 to align with the TRL and IRL scales.

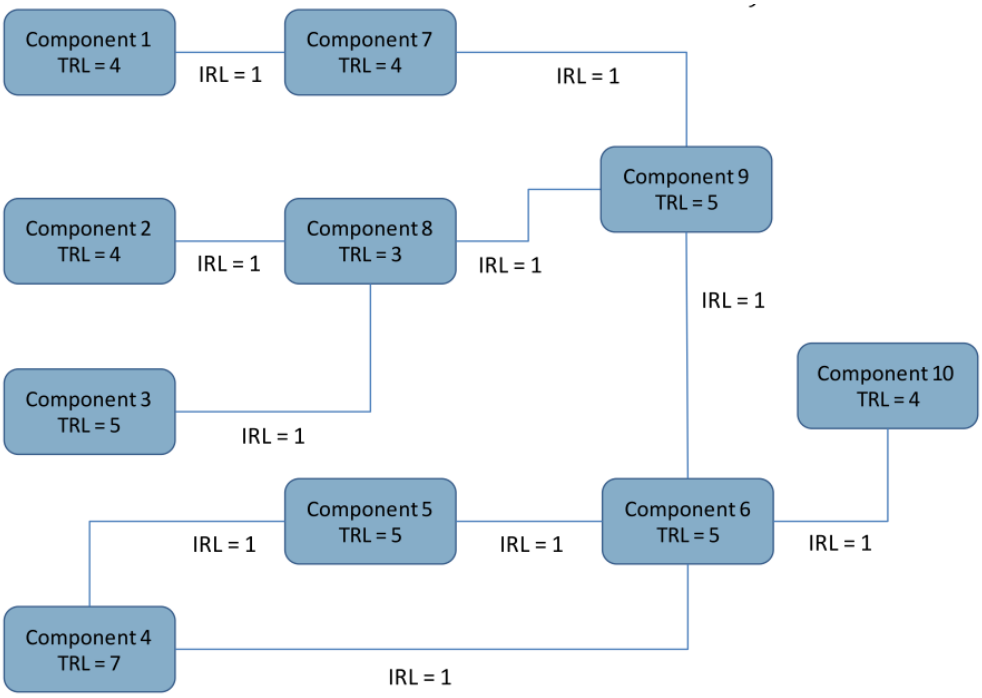


Figure 3.16: TRLs and IRLs from an Example 10-component System [9]

$$Composite\ SRL = \frac{\sum_{i=1}^{n} Component\ SRL_i}{n} \tag{3.8}$$

The composite SRL is the mean SRL, therefore in addition to the mean we also track the standard deviation of the component SRLs (SRL_σ) using (3.9) to understand the 'spread' of mature and immature technologies and integrations within the system. For example, two systems with the same composite SRL, or mean SRL, could have very different SRL_σ values. The one with a larger SRL_σ will have many more immature technologies than the one with a smaller SRL_σ. Additionally, tracking and reporting the minimum and maximum component SRLs by number/name provides another method of tracking specific program risk by most and least SRL risks.

$$SRL_\sigma = \sqrt{\frac{\sum_{i=1}^{n} (Component\ SRL_i - Composite\ SRL)^2}{n}} \tag{3.9}$$

This SRA process is accomplished each time there is an update to the system design and accomplished periodically to ensure the risks are being tracked accordingly. Tracking the SRA data

for systems during their early system development has shown these matured systems incur significantly less cost and schedule growth than systems that did not track their SRA metrics [116] and result in a more robust system [118]. Therefore, YAMM has adopted this assessment and refined it using SRL_σ for more information to feed to project management.

3.2 *Dev-V&V* Stage

In this section we define the YAMM *Dev-V&V* stage process. This is the second YAMM stage in the YAMM life cycle process as shown in Figure 3.1. We first provide some background into standard DevSecOps processes and challenges, environments, tests, and infrastructure to provide context for our novel methods of data collection and delivering system specifications and test cases to improve our modeling accuracy and enable solutions to some of the DevSecOps top challenges. The data is collected from Dev*Ops environments for empirically-based feedback into the UMA. Figure 3.17 illustrates this test data delivery to *Data Analytics & Algorithmics* as one output from this stage. Additionally, Dev*Ops continuous delivery of operational SW and DI to the *Operational Data System* stage as another output from this *Dev-V&V* stage is also shown in this figure. The third output from this stage, as shown in the figure, is system specific knowledge which comes in the form of the multidisciplinary team collaboration.

The analysis of test resource data and system metadata enables early system optimizations in the *MDAO* stage and increases SE system specific knowledge early in the system life cycle as previously discussed in Section 3.1. This test data also enables the development of the UMA automated V&V testing pipelines for the specifications and simulations also discussed in Section 3.1. In Section 3.2.1 we discuss the system specifications and test cases defined in the UMA as input into this stage for the DevSecOps specific environments and tests. These system specifications and test cases are illustrated in Figure 3.17 as the only inputs into this stage. In Section 3.2.2 we describe our data collection technique, the type of data collected, and an overview of its uses later in the YAMM life cycle.

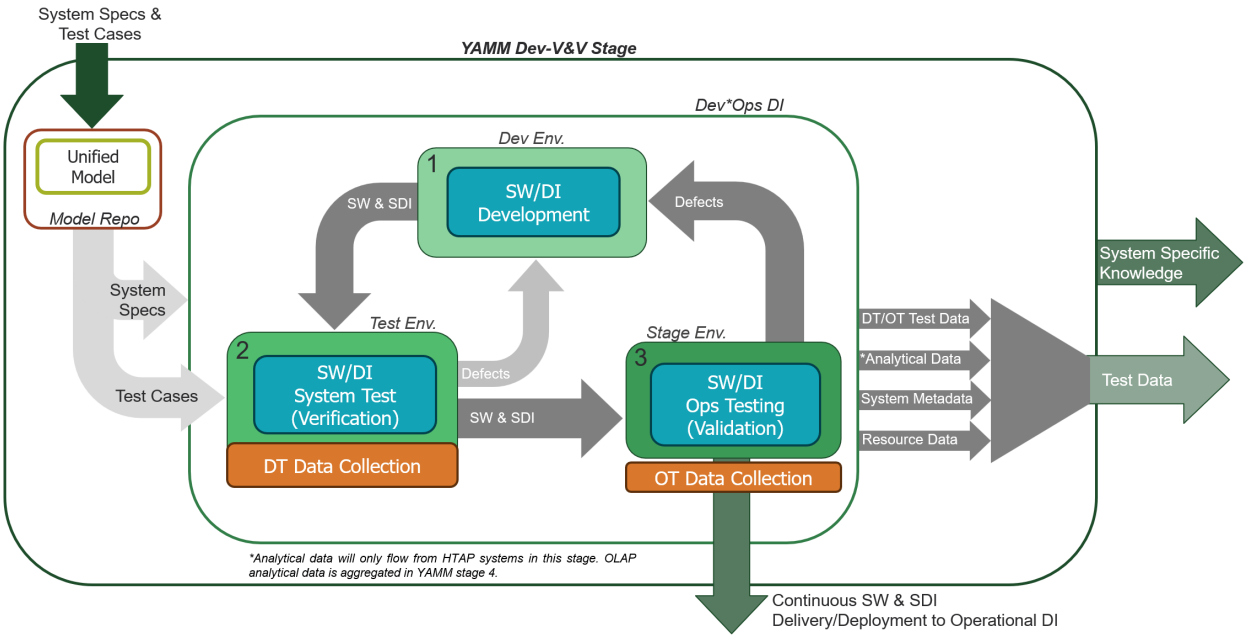


Figure 3.17: YAMM Dev*Ops & Digital Infrastructure Process

From Akbar et. al. [3] the definition of DevOps is a methodology that aims to establish collaboration between programmers and operators to automate the continuous delivery of new software to reduce the development life cycle and produce quality software. Furthermore, DevSecOps extends the DevOps concept, which integrates security methods into its process. This is a software development process where security is built in to ensure application confidentiality, integrity, and availability. We use Dev*Ops to denote either DevOps, DevSecOps, or additional automation added to these strategies. The top challenges of DevSecOps [3] are listed in Table 3.7, several of which we address with our system specifications and test cases delivered to this stage from the UM. We discuss this further in Section 3.2.1.

Figure 3.1 show the continuous delivery and/or deployment of new software as part of the DevSecOps process, however continuous unit, functional, system, and security tests must be written to enable this continuous delivery. This same figure shows three environments which are built to enable the this continuous delivery of software, namely the development, test, and staging environments. As the DevSecOps process moves software from steps 1 through 3, as shown in Figure 3.1, these three isolated environments increasingly restrict access and privileges to users to ensure the

Table 3.7: Top DevSecOps Challenges [3]

Challenge Number	DevSecOps Challenges
1	Lack of automated testing tools for security in DevOps
2	Ignorance in static testing for security due to lack of knowledge
3	Communicate security standard to the DevOps team
4	Lack of secure coding standards
5	Inconsistent security policy design and performance measures
6	Developers' resistance to integrating security protocols
7	Neglecting change control in security
8	Using unstable performance metrics for security evaluation
9	Translate compliance constraints to requirements
10	Lack of software security awareness
11	Integrate and deliver security features
12	Threat modeling scalability issue
13	Perform security feature review
14	Periodic penetration test for application coverage
15	Define secure deployment parameters and configurations
16	Identify software defects found in operations monitoring and feed them back to the development
17	Use application behavior monitoring and diagnostics
18	Use of immature automated deployment tools

integrity and validity of each stage's tests. This is also done to incrementally move software from a non-operational environment, like the dev environment at step 1, to step 3, the staging environment, which most closely represents the operational environment. This environment uses the same security controls, software versions, DI resource allocations, and everything else to catch any error before delivering software to the operational environment.

Each stage contains a set of automated tests that called DSOP that run tests appropriate for that stage. Additionally, each software movement from one stage to the next is accompanied by security scans which are also built into the DSOP. If any tests or security scans fail in the DSOP during any stage, then the software is not merged with the main software repository and the errors are sent back to the SW-Devs in the development environment to fix and start the process over.

The tests in each environment's DSOP are dependent on the target of the test and vary depending on the System Under Test (SUT), the conditions of the environment, and the budget/time devoted to the testing activities [4]. In Table 3.8 we list each of the four different testing levels,

as defined by the Software Engineering Book of Knowledge (SWEBOK) [4], and map each to one the three DevSecOps environments (dev, test, or stage) shown in Figure 3.17. We discuss the specifications of each environment and the test definitions from test cases in more detail in Section 3.2.1.

Table 3.8: DevSecOps Environments Mapped to Test Levels [4]

DevSecOps Environment	Testing Level	Test Level Definitions
Development	Unit	Verifies the functioning in isolation of SUT elements that are separately testable. These are individual subprograms, components, or subsystems.
Development & Test	Integration	Verifies the interaction among SUT elements such as components, modules, or subsystems. Integration strategies incremental integration of the SUT elements by MVP use case or capability. It may target different levels of integration and happen at different stages.
Test	System	Tests the behavior of the SUT which are not already tested by the unit and integration tests. It also assesses non-functional system requirements such as security, performance, reliability, etc.
Stage	Acceptance	Targets the deployment of a SUT with the goal to verify end users' expectations. Testing targets usability in an operational relevant environment.

In addition to software development and testing we provide context for the DI portion of data systems which builds the foundation of the three environments we just discussed. The development, test, and stage environments can share a DI as seen in Figure 3.17 as long as each of the environments are isolated such that tests within each environment are not affected by others. We define DI as the physical and software-based foundation that enables the exchange of digital information, products, and services. The components of a DI consist of, but are not limited to, physical Hardware (HW), cloud, fiber infrastructure, software, and cyber-physical security. The list below defines these terms.

1 **Physical HW:** Includes on-prem or aircraft HW

1.1 **Server hardware:** Physical equipment that stores and processes data

1.2 **LRUs:** Physical aircraft equipment that stores and processes data

1.3 **LAN:** Local on-prem physical networks that transmit data

1.4 **Avionics Bus:** Local aircraft physical networks that transmit data

2 **Cloud:** A service that provides storage, compute, and network resources

3 **Internet Infrastructure:** Open Systems Interconnection (OSI) layers 1-4 networks that transmit data between cloud and on-prem

4 **Software:** Includes operating systems, virtualization, and applications

5 **Security:** Includes cyber-physical systems to protect data, networks, HW, SW, and physical spaces

Figure 3.18 is a visual representation of the components needed to manage an on-prem data center, create infrastructure in an IaaS, run applications on a PaaS, or consume applications through SaaS. This structure is similar for aircraft or other data systems. Garrison [10] defines cloud native infrastructure as infrastructure that is hidden behind useful abstractions, controlled by APIs, managed by software, and has the purpose of running applications. Running infrastructure with these traits gives rise to a new pattern for managing that infrastructure in a scalable, efficient way. We see a rise in application of cloud native infrastructure in aircraft and on-prem servers, therefore a cloud native infrastructure is not only limited to the cloud.

The cloud native infrastructure (IaaS) shown in Figure 3.18 is software defined using different tools and Infrastructure as Code (IaC). This is software defines the amount of compute and and storage your system or sub-system can use and also defines the Virtual Network (vNet) connecting your virtual resources. This software defined DI, also known as SDI, is the focus of our data collection efforts for resource data and system metadata. This data is collected in the test and stage environments as shown in Figure 3.17 and is then sent to the YAMM *Data Analytics & Algorithmics* stage.

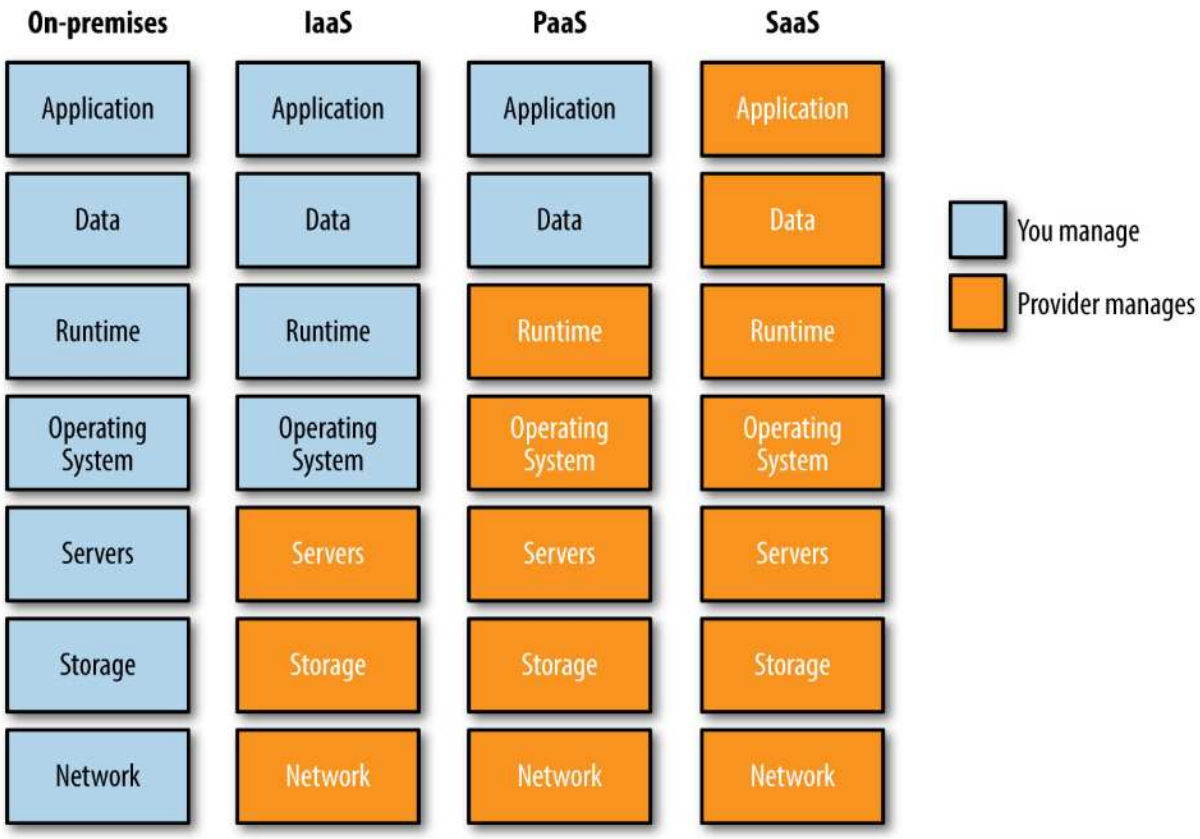


Figure 3.18: Cloud Native Infrastructure Layers [10]

Resource data and system metadata collected on SDI allows analysts to understand the state of the system resources as a function of time, which enables the creation of algorithms to capture the behavior of this virtual DI. These algorithms are the core of building an accurate UM that represents the system behavior. For cloud applications understanding their resource requirements for their use cases is important to understand the cloud resource costs associated. For on-prem applications running on a SDI it is important to understand their resource requirements to be able to forecast acquisition budgets of new on-prem resources and spaces as needs grow. We discuss the data collection techniques further in Section 3.2.2.

3.2.1 DevSecOps System Specs & Test Cases

In this section we discuss the system specifications and test cases that must be built into the UM and presented to the multidisciplinary SW-Dev and DI teams in this stage. We define the

requirements to specify, not only the operational system, but also the system that supports the development and test of the operational system. Additionally, we define the need for specifications will enable security solutions to the DevSecOps challenges listed in Table 3.7. We also list the need for test cases that define system tests to be accomplished in the test environment shown in Figure 3.17.

The operational system specifications are delivered to the multidisciplinary DevSecOps and DI teams in context of structural and behavioral requirements. This enables them to decompose these requirements into lower level sub-system requirements, unit tests, and integration tests. They design and develop the sub-systems to satisfy these system requirements. The tests they develop verify the sub-system requirements from within the development environment. Furthermore, test cases passed to these multidisciplinary teams are design and developed to verify the system requirements in the test environment.

However, before the development and testing of the operational system can begin the development and test environments and DI shown in Figure 3.17 must be designed and built. In addition to the operational system specifications, the system to develop and test the operational system requires environment specifications and needs to be defined in the UM. These specifications include the size, performance, and type of resources that will support the development and testing activities mentioned above. These resource requirements are fed to the DI team responsible for developing the SDI. In addition to operational system security policy specifications the UM needs to specify security controls and policies for the development, test, and staging environments.

The aspect of security in DevSecOps faces many challenges as shown in Table 3.7. If these security challenges are not addressed at the system level, then sub-system requirements will be absent as well. Without these security specifications the operational system be vulnerable to attacks which can cost organizations millions of dollars and delay schedules. Many of these challenges listed in Table 3.7 can be addressed through system security specifications that we cover in the next several paragraphs.

DevSecOps challenges 1, 2, & 14 in 3.7 are related to security testing and require system specifications and test cases from the UM that provide additional knowledge of the types of required static security tests or penetration tests.

DevSecOps challenges 3, 4, 9, 11, 12, 15, & 17 relate to security standards, requirements, security design, and software behavior monitoring therefore the UM must define the organization's security policy, break them into requirements, secure coding standards, secure design patterns, secure SDI environments, secure physical DI, and operational application monitoring and diagnostic requirements. Test cases involving system level security policies are also required to be defined in the UM to help solve these issues.

DevSecOps challenges 5, 6, 7, & 8 relate to compliance and policy specifications. Security policies and performance measures must be included in the system specifications, however very little can be done in terms of specifications that will enable solutions to developers' resistance and neglect for implementing best practices. Sanchez-Gordon and Colomo-Palacios [120] site developers main resistance to security policy implementation can be attributed to the focus on meeting the delivery schedule. This is an area where multiple objectives such as security, schedule, and cost are opposing. To help solve this issue these objectives need to be quantified via security's impact to schedule and cost, then the system and it's schedule needs to be optimized and analyzed within the YAMM *MDAO* stage.

We move on to a discussion of the uses of test cases developed in the UMA. The test cases developed within the UMA are focused on system level and some integration levels tests. These tests verify the functional and non-functional requirements and system level use cases.

In addition to this typical function of test case, we also define out novel method of defining test cases that validate the model simulation. These test cases ideally serve three functions, 1) verify the system use case behavior and specifications, 2) validate the simulation behaviors, functions, and utilities allocated to the system use case being tested, and 3) develop empirically-based algorithms to represent the system behavior in the UM. The UM specifies the type of data to collect for each test case, the tagging method for the test case data, and the method to store the data to

make it available for the UMA to validate the model. Section 3.2.2 describes the methods of data collection.

3.2.2 Test Environment Data Collection

In this section we discuss the which environment in Figure 3.17 collects each type of data listed in Tables 3.1 and 3.2 and methods of recording stochastic data for the purpose of developing algorithms.

In addition to resource data and system metadata Figure 3.17 shows the collection of DT and OT test data as well as analytical data. However, analytical data is only collected in this stage if the data system being designed is an HTAP system which updates analytical data in real time. The type of data collected for each data system category is listed in Tables 3.1 and 3.2. To clarify this table in context of our DevSecOps stage, if this system is an OLTP system then operational data is aggregated in a separate OLAP system for analysis, where the analytical data is at rest. Therefore if this is an OLTP system, then only DT and OT test data, as representative operational data, is sent from this V&V stage to the YAMM *Data Analytics & Algorithmics* stage. This stage contains the associated OLAP system.

However, if the system under design and development is an OLAP system, then we define system analytical data as meta-analytical data which informs analysts about the operation of the OLAP analytical system. This could keep going to multiple versions of meta-meta-analytical data, however we do not keep going down that proverbial rabbit hole of meta-metadata and simply store this meta-analytical data in the same OLAP system for analysis in the YAMM *Data Analytics & Algorithmics* stage.

A data collection framework is setup in tandem with the testing framework in order to automatically collect the right data. Two types of data collection are needed with one that is associated with a testing framework to collect statistically significant data, while the other type of data collection must operate passively to collect system end-user behavior and the system performance and resources associated with this. The data collection framework tied to the testing framework is

only setup in the test environment shown in 3.17, whereas the passive data collections framework is deployed to the stage and operational environments. This separation is done to ensure the stage environment resembles the operational environment as closely as possible.

In terms of the data collection and testing framework in the test environment it must be setup to quantify application performance and resource usage for the test cases defined from the UM. Quantifying the performance/resources of different applications, through data collections, in a shared resource SDI under transient conditions is extremely challenging [58] for the following reasons: 1) Shared resources exhibit stochastic resource capabilities such as supplies, sizes, and varying demands, 2) Users have heterogeneous, dynamic and competing Quality of Service requirements, 3) Applications also have heterogeneous performance, workload, and dynamic scaling requirements. Therefore, we use a statistically significant amount of SDI performance and cost data to provide a distribution of results in which to validate the model and train algorithms.

The type of data and the technique of collecting this DT and OT resource data is a crucial step in ensuring the success of modeling the SDI or cloud-based system performance, resources, and costs later. For instance, the behavior of SDI-based software has unpredictable performance fluctuations caused by *performance uncertainty* [121] and recording data for a single software execution event is not sufficient to properly capture generalized performance and resources of that software. Therefore, it is necessary to apply a statistics-based performance and resource testing methodology to capture enough data to confidently estimate the distribution. PT4Cloud is one example of implementation in a test environment and results in a 95.4% accuracy on average while reducing the number of test runs by 62% when compared to extensive performance and resource tests. The reduction of resource consumption during the test runs by this methodology is especially important to consider when trying to reduce costs and resources of the system.

In the PT4Cloud methodology the test activity executes the SW performance tests repeatedly over a specified time interval and records the performance and resource results. A second test executes the same performance tests repeatedly over the amount of time on a different date and aggregates this data with results from the first test. The performance distributions for the previous

data set (D_{n-1}) and the newly updated data set (D_n) are compared using the Kullback-Leibler Divergence (KL-Divergence) equation shown in (3.10).

$$D_{KL}(P||Q) = \int P(x) log \frac{P(x)}{Q(x)} dx \tag{3.10}$$

Where P and Q are two distributions, such as D_{n-1} and D_n, over a random variable x and $D_{KL}(P||Q)$ is the KL-Divergence where a value of 0 equals no divergence and a value of ∞ indicates the distributions are completely different. However, divergence is non-intuitive, therefore likelihood and probability are calculated by (3.11) and (3.12), respectively.

$$L(P||Q) = 2^{-D_{KL}(P||Q)} \tag{3.11}$$

$$P_s = L(D_n||D_{n-1}) X L(D_n||D_{n-1}) \tag{3.12}$$

Where P_s is the statistical probability that both distributions are equal. If the statistical probability P_s is less than the objective probability P_o then the test and data collection framework begins another software performance testing interval. This iteration continues until the P_s is equal to or greater than the P_o, at which point a confidence band is calculated for the final performance distribution. The software tests must be performed on multiple variations of SDI resources such as: VMs, cloud storage levels, and egress capabilities to provide the information needed for the optimization algorithm. The data collected is used for algorithm training to provide higher fidelity simulation capability during the optimization methodology and to validate the UM simulation capabilities.

3.3 *Operational Data System* Stage

This section briefly describes the delivery and deployment of the software and DI from the YAMM *Dev-V&V* stage as well as the operational data collection method to support the development of algorithms and the UM in YAMM stages four and one, respectively. The deployment of

software applications and the SDI happen continuously within the DevSecOps framework. This deployment is shown in Figure 3.19. The hardware within the DI happens on a slower cycle and must be designed into the delivery schedule.

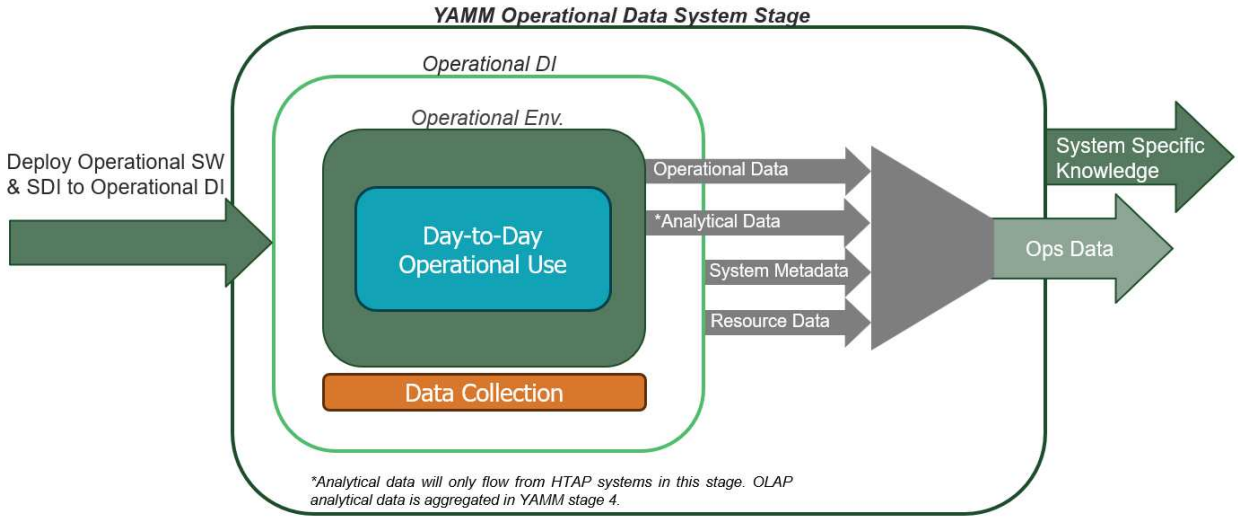


Figure 3.19: YAMM Operational Data System Stage

3.3.1 Operational Data Collection

For the operational environment a passive data collection framework must be used, this type of framework was described in 3.2.2 for the stage environment. The purpose of this is to collect operational data from this OLTP or HTAP type system which consists of end-user transactions, real time processes, usage patterns, performance metrics, and failures or warnings. If this is an HTAP system then this operational data will be aggregated in real time and this analytical data will be collected as well. These real time operational and analytical actions will consume resources and this resource data will also be collected. Finally, the system metadata will be collected and sent with all other data collected to the YAMM *Data Analytics & Algorithmics* stage for analysis and algorithm development.

If the system under design and development is an OLAP system, then we refer to the operational data collected as meta-analytical data since it is data about the operational use of an analytical system. This data is also collected and passed to the next YAMM stage.

Each CSP has its own passive data collection framework which provides its customers with reports on resource usage, costs, service charges, and any other support charges. Using these services with CSPs is recommended and most likely the best source of data you can get in your cloud-based operational environment without modifying it with a testing framework mentioned in Section 3.2.2. However, it is important that the software developers tag each appropriate application and service in use in order to properly classify each resource and cost. If the organization has plans to move operations from the cloud to an on-prem system, then developing a data collection framework is necessary for the on-prem system.

This operational data will provide valuable insights into possible non-Gaussian distributions depending on the way the users interact with the system. Additionally, new usage and behavior trends may be present in this data which informs new tests and new behaviors to model. Tagging or storing operational, OT, and DT data separately will ensure these insights remain clear.

If implementing an operational analytics system (HTAP) like a Data Mesh architecture which is a socio-technical architecture, the responsibility of operational and analytical collection is assigned to the Data Domain owner and the Data Product developers. For YAMM we took this a step further and extended the data collection responsibilities of the Data Domain owner to include resource and system metadata.

The focus of this YAMM *Operational Data System* stage is to ensure data collection and the data delivery to the *Data Analytics & Algorithmics* stage is automated. The operational environment will have the most realistic data, but it will also most likely be the most sparse.

3.4 *Data Analytics & Algorithmics* Stage

In this section we discuss the last YAMM stage and its process which analyzes DT, OT, and analytics data and system metadata, trains and selects algorithms on this data, and finally delivers

these results back to the first YAMM *MDAO* stage as shown as output in Figure 3.20. This begins the YAMM incremental and iterative process for a new MVP or an update to the UM. This figure also shows the UM specifications, test data, and operational data inputs into this stage as discussed in Sections 3.1, 3.2 and 3.3, respectively.

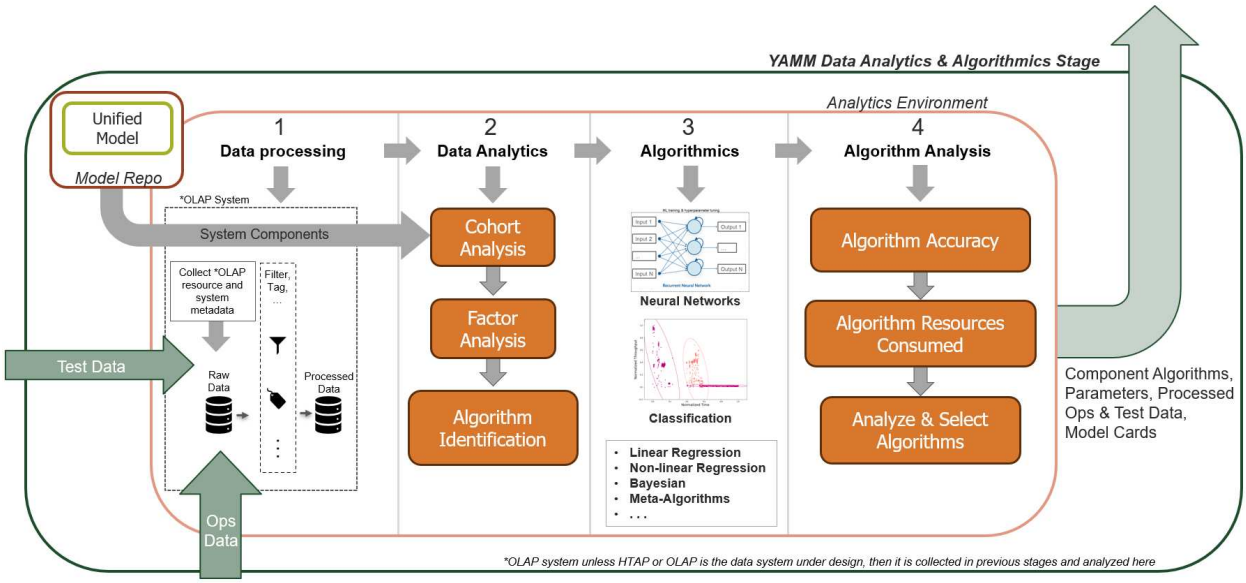


Figure 3.20: YAMM Data Analytics & Algorithmics Process

The first step of this stage is to ingest operational and test (OT & DT) data from the data collection framework. If the data system under design is an OLTP system, then this step represents the associated OLAP system aggregating the real time operational data from the OLTP system. If this is the case, this OLAP system in this stage also collects its own resource and system metadata to analyze. If the data system under design is an OLAP or HTAP system then this stage is just an extension of that system and processes data that has already been aggregated. In parallel with data aggregation, step 1 also ingests the data system components' information from the latest UM version and prepares this information for the next step.

Step 2 begins with a cohort analysis on the operational and test data, however this analysis is initially informed by the data system components' information. Cohort analysis groups components with shared characteristics to identify trends and patterns of those groups over time. Cohorts

are grouped by shared characteristics like types of storage, compute, networking, software, and data format to name a few. This helps the data science team in this stage understand how each group's behavior, performance, resource usage, or end-user usage patterns change over time. The operational and test data are split into these cohort groups to map recorded behavior and resources to system components. The system metadata collected throughout this process should enable this mapping of data to UM components through shared metadata information.

Once the operational and test data are sorted into system component groups by the cohort analysis we apply a factor analysis to each group to help identify patterns and relationships within this complex data. This analysis aims to uncover latent structure underlying the set of observed data which identifies specific parameters or latent variables that explain correlation in the observed data. These correlations provide insight into how to model the simulation logic within these cohort groups by using the latent variables as inputs for the UM. The correlations within the data from this factor analysis also informs the data science team towards identification of which algorithm best models each system component cohort group. For example, algorithm identification analyzes the data to model and decides if linear or non-linear algorithms need to be used or decides if it is a classification, Bayesian, or complex regression problem that needs ML to model. We discuss additional numerical methods for simplifying complex mathematical problems using parametrics and/or discretizations in Section 3.4.1.

Figure 3.21 illustrates our general algorithm development process. The top half of this figure represents the cohort group data analysis process that starts in step 2 after the cohort groups have been created. Starting with the data from each cohort group our process moves through different layers and iterations of data filters, factor analyses, and algorithm identification to produce the processed data, observed correlations, latent variables, and algorithms to train in step 3 of the process shown in Figure 3.20.

Step 3 ingests the results from step 2 which are sets of data cohort groups mapped to system components each with observed correlations, latent variables identified, and a list of algorithms identified to best model the data. We then divide each cohort group of data into three parts; training

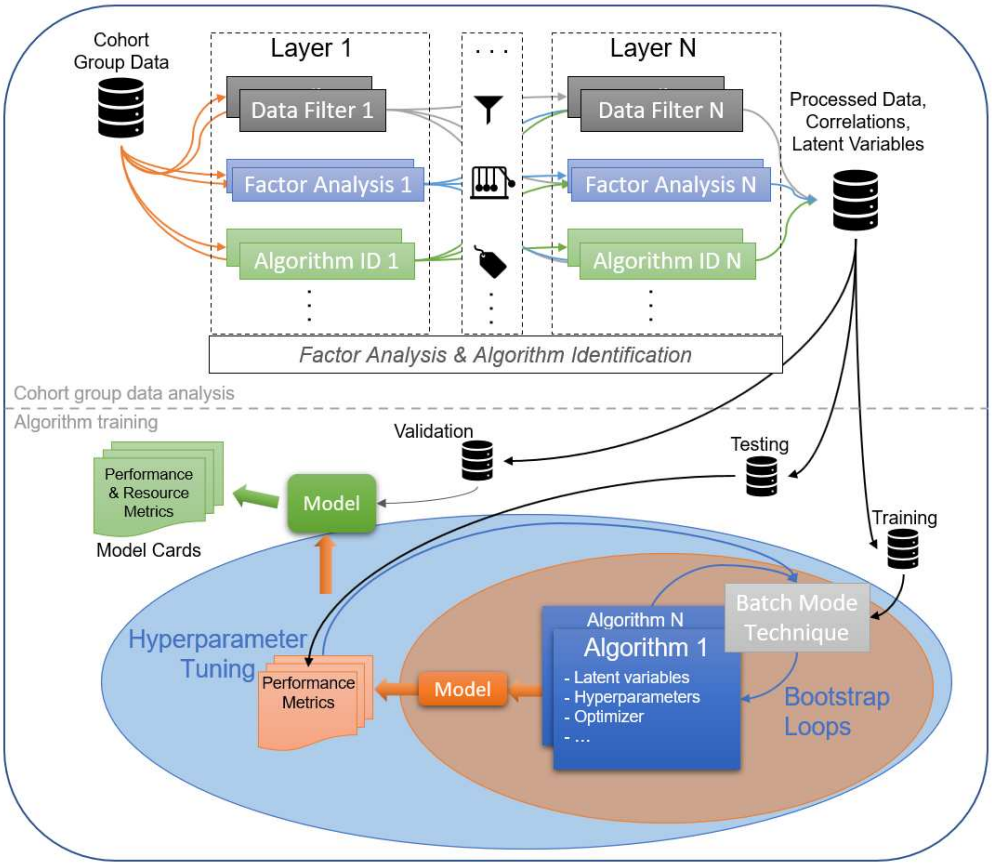


Figure 3.21: General Algorithm Training, Testing, and Validation Process

data, testing data, and validation data as shown in the bottom half of Figure 3.21. Training data is used to train the list of algorithms identified for each cohort group. The testing data is used to measure the algorithm residuals as they are converging on an answer. This convergence is usually associated with the ML category of algorithms during hyperparameter tuning. The validation data is used in the next step to measure each algorithm's accuracy and resource usage. We train more than one algorithm on this data so that we can identify the best one or the best combination of several algorithms in the next step. We discuss more about combining many algorithms into a single algorithm, or meta-algorithm, in Section 3.4.2.

Step 4 shown in 3.20 uses the validation data to measure the accuracy of each set of algorithms within each cohort group. Each algorithm's accuracy is recorded into its Model Card along with a pointer to the training, testing, and validation data to provide documentation for each algorithm model and for the algorithm analysis and selection process later in this stage. We measure the per-

formance and resources used for each algorithm as its predictions or inferences are being validated with the validation data set. The performance metrics are prediction accuracy and speed, while the resource consumption metrics are compute cycles, volatile memory usage, non-volatile memory usage, and I/O bandwidth consumption. This resource consumption and performance is written to each algorithm's Model Card shown in Figure 3.21. The compute resources used to measure the performance must be the same as the compute resources of the system that is running it. For example, if the algorithm is planned to be run in a simulation in the cloud on a specific VM hosted on specific HW, then that VM and HW must be used to measure the algorithm's performance. If not, the results of the algorithm analysis may be skewed. Finally, for step 4, algorithm Model Cards are used to analyze and select the best algorithm or meta-algorithm using the performance and resource usage measurements as the metrics for the decision.

These new component algorithms, parameters, latent variables, Model Cards and the processed data is sent to the first YAMM stage to be incorporated into the system's UM.

Section 3.4.1 describes different methods of modeling time dependent and complex systems through the use estimating complex behaviors with simple parametric equations. Additionally, it discusses another numerical method in which the physical or temporal system space being modeled is broken into discrete parts to be able to estimate complex system emergent behaviors.

Section 3.4.2 discusses a philosophy and its principles for training algorithms. The general idea is that using many simple algorithms to model complex systems is more accurate and robust than a single complex algorithm to model the same system. This philosophy is known as meta-algorithmics.

Section 3.4.3 discusses the automation of training ML models to be deployed to an operational environment and how we adapt this to enable this capability to deliver algorithms or meta-algorithms to the *MDAO* stage for use in the UM.

Finally, in Section 3.4.3.1 we provide additional detail about the algorithm analysis and selection process based on performance, accuracy, and resource usage.

3.4.1 Parameterization & Discretization

Numerical methods, such as parameterization and discretization, provide engineers with a way to formulate complex mathematical problems so that they can be solved with more simple arithmetic operations. The Unified Method uses time discretization to calculate the data system's time dependent behavior within smaller discrete time slices, making it easier to capture critical time dependent system emergent behavior. These emergent behaviors may not be present when estimating a single total cost or performance metric since many emergent behaviors in data systems rely on order of operations and timing. The concept behind the numerical method of discretization also encourages the domain to be broken up into discrete physical components. The CFD example in Section 2.4.2 is an advanced example of discretization that breaks the physical space of the fluid into discrete finite volumes or finite elements, then solves a set of finite-volume or finite-element equations derived from the governing fluid flow equations, known as the Navier-Stokes equations.

With the UMA, we discretize, or decompose, the data system components into logical objects. Since there are no governing equations for software, we must estimate each component's behavior in terms of data in, data out, stored data, removed data, and work done on data. Work done on the data is a function of compute, network, CRUD, and any other resource usage and cost associate with software acting on data. We can estimate these component behaviors through different modeling and numerical approaches. An early approach is to parameterize the behavior which involves fitting the data with a mathematical function that best estimates the dependent variables provided the independent variables. This can also be done through factor analysis which identifies correlations in observed data and their latent variables for empirically derived problems without mathematical functions. These mathematical functions can be as simple as the arithmetic mean to as complex as a full DL model. YAMM starts with arithmetic means for initial estimates early in the data system's life cycle when data collection is simple and is usally a rough estimate from the software development or test environments and our trust in its accuracy is low.

Once a suitable model is trained or parameterized using empirical data, it is incorporated into the component of the UM that the data represents. By parameterizing individual components of

this system it also allows reuse of that component in different use cases. Further exploration into Numerical Methods for Engineers [122] is recommended for new applications of YAMM.

3.4.2 Meta-Algorithmics

The purpose of this section is to discuss the overall YAMM data analytics philosophy rather than provide an overview of the broad field of data science towards modeling system behavior. Within YAMM we adopt the meta-algorithm philosophy discussed in Meta-Algorithmics [72] recommending that "instead of finding the best possible intelligent algorithm, system, or engine for a task, the system architect should look for the best combination of algorithms, systems, or engines to provide the best accuracy, robustness, adaptability, cost, and so on." Meta-algorithms are algorithms that operate on other algorithms or combination of algorithms, often combining them in some way.

Since emergent system behavior is complex to model and can appear to be stochastic, we decompose the system into smaller, logical components that are easier to model, then model these components using whichever algorithm or combination of algorithms match best. In this approach, the UM becomes the core mechanism that intelligently combines each algorithm to best estimate the complex system emergent behaviors. The list below lists Simske's *eight principles of meta-algorithmics* [72] that YAMM adopts with a combination of system and architectural models and empirically-based algorithms.

Principle 1: No single generator—algorithm, system, or engine—encapsulates the complexity of the most challenging problems in artificial intelligence, such as advanced machine learning, machine vision and intelligence, and dynamic biometric tasks. (i.e. A plurality of algorithms is more likely to be able to provide a correct answer, from at least one of the algorithms, than a single algorithm is.)

Principle 2: It makes sense for a system designer to optimize an algorithm for a portion of the input range and then leave the rest of the range to another algorithm.

Principle 3: Patterns of usage are often more powerful—more accurate, more robust, more trainable, more reusable, and less costly—than a single, highly trained algorithm, system, or engine.

Principle 4: Ground truthing, or the labeling of training data, is extremely expensive—in general, we have to assume that there will be a relative sparseness of training data.

Principle 5: First-, second-, and third-order meta-algorithmics are used to create a highly trainable system with minimum cost of on-ramp. Commercial off-the-shelf, open source, and targeted generators can be deployed together in these patterns. The targeted generators are designed to provide good results for partitions of the input space on which the existing generators do not perform acceptably.

Principle 6: The juxtaposition of bottom-up and top-down algorithms, and the combination of targeted and broad algorithms can be used to generate systems that are resilient to changes in the input data, and highly adaptable to subsystem deployment.

Principle 7: Weighting and confidence values must be built throughout the system in order to provide the means to use meta-algorithmics on multiple classifiers at a time; hybridize the multiple classifiers in a plurality of ways; and/or as an alternate to parallelism by task and parallelism by component. Meta-algorithmics, in this way, provide the means for simultaneously combining, learning, and parallel processing.

Principle 8: The goals of modern algorithm designers will be, increasingly, indistinguishable from the goals of modern system designers. Architecting for intelligence is becoming a primary need for architecting any system, with the increasing pervasiveness of big data, analytics, voice search, and other machine-to-machine intelligent systems.

In the two case study applications of YAMM, we explore the strengths and weaknesses of several individual algorithms and combined algorithms for data modeling that include MBSE,

machine learning, data mining, classification, regression, parameterization, and system analysis towards estimating data system performance and cost.

The approaches discussed in this paper are not all inclusive and we recommend further exploring Patterns of Intelligent Systems which describe the design patterns and first, second, and third order meta-algorithms listed in the principles above [72]. Furthermore, patterns of parameterization and discretization, as a subset of meta-algorithmics, are the foundation that the UMA is based on. This is discussed in Section 3.4.1.

3.4.3 MLOps

Ideally, for large data systems that need ML to drive their UM they would develop an MLOps pipeline [123] for each system component, however the brutal reality is that a large number of ML project fail to ever make it into production [124]. Therefore it is best to balance the complexity of the component you are trying to model vs the amount of effort it will take to develop that model. In some cases a full MLOps pipeline is more work than the effort to decompose a complex component into several simple components within second order or linear models and combining these components into the UM. According to to Meulen and Gartner [124] "In general, it's best to start AI projects with a small scope and aim for 'soft' outcomes, such as process improvements, customer satisfaction or financial benchmarking."

With this warning about putting too much effort into single algorithms, thus not adhering to Meta-Algorithmic principle 3.4.2 above, we review the concept of ML algorithms. The following sections discuss our automated approach to ML training hyperparameter tuning, and selection using representative test or operational data followed by an automated scoring and analysis approach using system resource and costs as part of the metrics to choose the best ML model. This ML selection and analysis approach can be used with YAMM, (i.e. these methods are not a requirement of YAMM), rather they provide an example of the philosophy of YAMM to automate and collaborate as much as possible to bring operations into the collaboration between software developer and system engineer.

Typically, any lengthy approach to train large ML, DL, or ensemble models on data should be implemented after sufficient representative test or operational performance and cost data has been collected to capture the behavior of the system for modeling purposes, otherwise this may be a waste on resources if you don't have sufficient data or representative data. This approach can also be implemented for systems that would implement ML models as part of their functionality, such as the aircraft sensor data system case study presented later. Therefore, there must also be enough data to train the functional ML model (e.g. sensor data for sensor fusion, or other software components of a hybrid cloud system).

3.4.3.1 Algorithm Selection and Analysis

Investigative analysis on the data (performance, cost, or other objectives/capabilities) provides key characteristics that inform the selection of the most fitting algorithms for the problem and the best preprocessing techniques for the raw data that is recorded. It isn't always clear which combination of these algorithms and preprocessing techniques will generate the best model for each data system component. Additionally, within the context of ML there are different methods of training that can affect the accuracy, resource usage, and cost of training and running the models, therefore all of these variables must be considered in order to automate the training, selection, and analysis of an ML model.

In an attempt to automate the training, analysis, and selection process and to arrive at a good solutions quickly we created the Gradient Descent Multi-Algorithm Grid Search (GD-MAGS) approach [26] and supplemented it with our extensions to ML Model Cards [15]. GD-MAGS is an optimization selection approach for ML and preprocessing techniques.

The algorithm analysis is focused on the system in which the algorithm will be running and is system specific. Therefore, the application of an Analysis of Alternatives (AoA) is performed for each of the different algorithms and their use cases on the system. The AoA objective function is a function of algorithm accuracy, speed, latency, memory used, processor cycles, and I/O bandwidth. All of these values for each algorithm is present in that algorithm's Model Card. Alternatively, a Monte Carlo analysis can return the distribution of possible objective function values given a range

of random use cases and use case variables. The results of these analyses enable the selection of the best algorithm for the system.

Chapter 4

Case Study 1: Legacy Aircraft Sensor Data System

The objectives of applying YAMM to this legacy aircraft data system are to:

1 Demonstrate YAMM on a real-world legacy data system example to provide user insights into the methodology better than discussing its philosophies and general principles alone

2 Demonstrate the value of collecting operational data and automating empirically-based algorithm training and selection

3 Demonstrate YAMM's flexibility to train ML models for use in an OLTP data system

4 Demonstrate the value of YAMM's DevOps capabilities to integrate OLTP software development with system engineering system analyses on resource-limited data systems

5 Demonstrate an example of applying meta-algorithmics principles on a complex ML modeling problem such as sensor fusion

4.1 Case Study Overview

An illustration of this legacy aircraft case study is shown in Figure 4.1 where the aircraft contains one to many sensors, one-to-many LRUs, two Multiplexed (MUXs) buses, and a MUX bus data recorder. This aircraft sensor data system is considered an OLTP system that has real time transactional processes with each sensor and LRU enabling the aircraft to function. For this case study there is no functional OLAP system to aggregate its data for the purposes of analysis. In our experience with the United States Air Force (USAF) we have noticed the lack of functional OLAP systems focused on legacy aircraft avionics and sensor data.

The application of YAMM to this legacy aircraft will following the portions of life cycle shown in Figure 4.2. This case study could not include deploying ML software to an aircraft so, instead, the focus will be on stages 3 and 4, which are the YAMM extensions. These novel aspects include the collection of empirical data the *Operational Data System* stage which is sent to the *Data Analytics & Algorithmics* stage for ML training, validation, and Model Card creation. After the ML

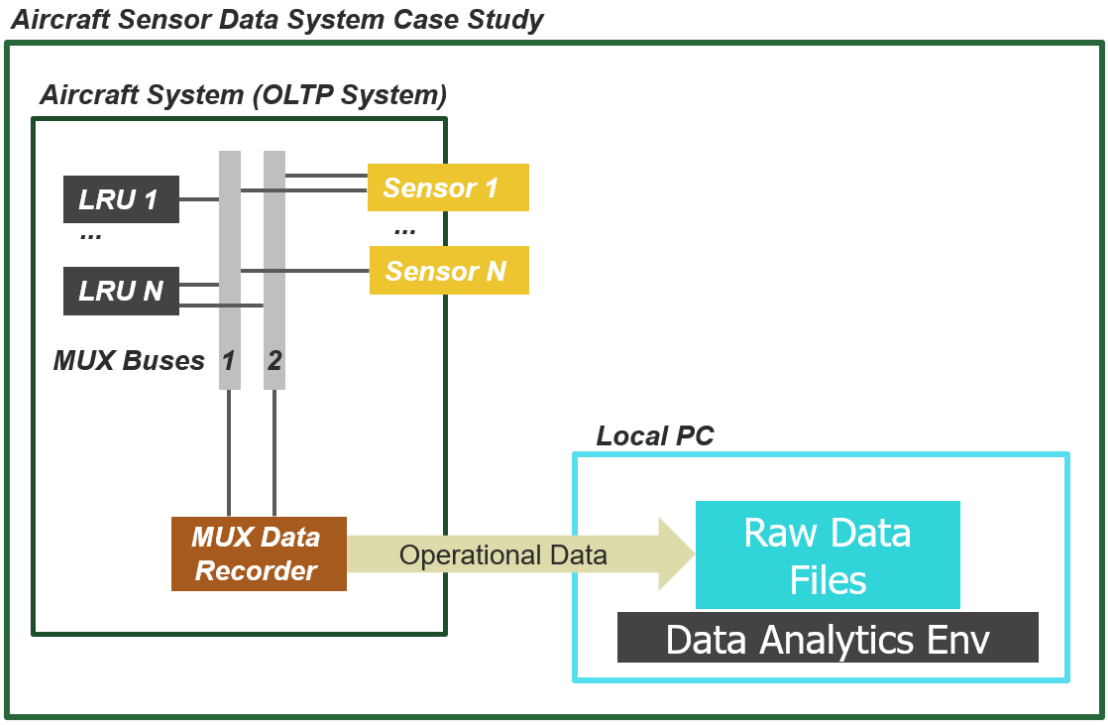


Figure 4.1: Legacy Aircraft Case Study Illustration

models are trained and ML Model Cards created, they will be will be sent to *MDAO* for system analysis and ML model selection. The final ML model selection is sent to the *Dev-V&V* stage for software implementation, testing, and integration. Figure 4.2 represents sections of the YAMM life cycle that are being exercised towards this chapter's objectives. These objectives are focused on the automation of 'closing the operational data feedback loop'. We removed the data flows from the figure that will not be exercised in this case study.

Multi-sensor data systems such as aircraft and autonomous vehicles use a suite of multi-modal sensors to collect data about their environment. A sensor fusion algorithm combines the sensor data into an integrated solution that represents the environment more accurately than what could be accomplished using a single sensor. Sensor fusion and sensor fusion architectures have been researched for many decades [125–128] and continuous advancements have been made as sensors, computational power, fusion algorithms, and the collection of data improve in performance and capability. Currently, one of the fastest growing segments in sensor fusion is in ML and DL algorithms. DL is becoming the leading approach compared to previous classical methods in time-

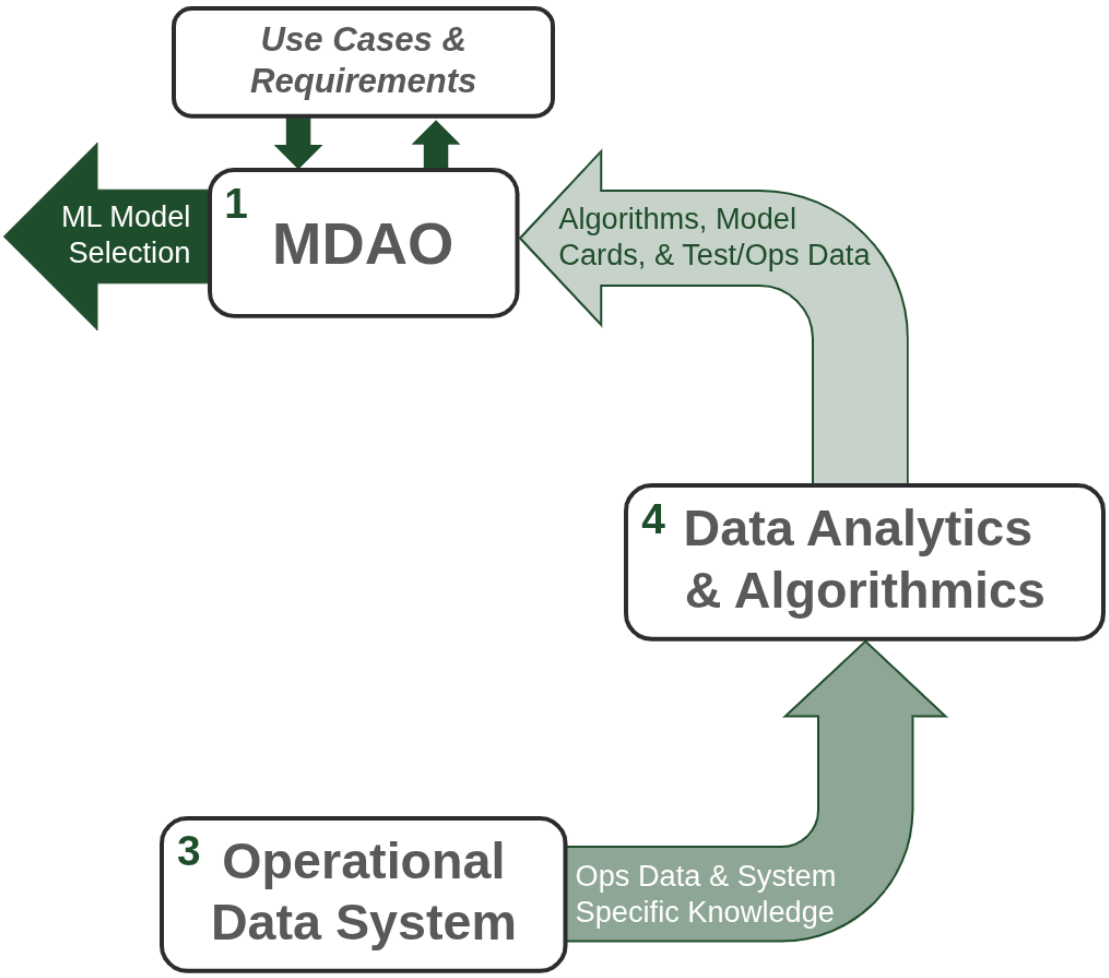


Figure 4.2: YAMM Life Cycle Stages Demonstrated for the Aircraft Case Study Example

series prediction algorithms as shown by the performance of RNN based approaches in the M5 competition [81]. These large ensemble DL algorithms consume large amounts of computational resources when forecasting. This is not a problem with scalable cloud-based compute resources, but the ability to run these models on a resource-limited system, such as aircraft or automobiles, may not meet system timing requirements. When considering sensor fusion solutions for resource-limited systems, choosing the best sensor fusion method is more than selecting the model with highest accuracy and precision [129]. Sensor system software designs need to estimate the resource needs of new sensor fusion models as they are developed and before they are implemented. The speed of developing software and new sensor fusion models will continue to outpace hardware

upgrades. Therefore, system analysis and design of these limited-resource systems must keep pace with rapid releases of new DL sensor fusion architectures, algorithms, and models.

Safety of flight, system robustness, security, openness, and system modularity objectives are considerations for system analysis and design. MBSE has been shown to provide significant advantages to project performance [60,98] for complex systems. The system development, qualification, verification and validation, and trade studies were more efficient while using MBSE. SysML has also become the ubiquitous MBSE language in the aerospace industry and if these SysML system models become the ASOT for aircraft system specifications, then it is imperative that these SysML models contain the information needed for sensor system analyses. This information needs to delivered in a standardized machine-readable format or API for future analysis scalability. In addition to the need for a standardized SysML system model API, DL models need to have machine-readable data, or Model Cards [95], to enable the automation of and to integrate sensor fusion DL algorithm development with system analysis and design (i.e. close the Ops feedback loop). These ML Model Cards need to contain quantitative data of the sensor fusion performance (accuracy, precision, reliability, latency, etc) and the resource costs (CPU cycles/speed, volatile and non-volatile memory, data size and rates, etc) of the system that will be using the ML models for predictions. This is an example of the need of YAMM's DevOps extension to the Harmony aMBSE.

Multi-modal sensor fusion presents numerous challenges that arise from the data to be fused, the imperfection and diversity of the sensor technologies, and the nature of the environment. Of the many challenges listed in Khaleghi et al. [130], this approach addresses the following: outliers and spurious data, conflicting data, data imperfection, data modality, operational timing, and static vs dynamic phenomenon. For example, in this case study, variables such as velocities, accelerations, pressures, temperatures, and aircraft orientation and position are fused to calculate the aircraft's altitude. In practice, certified altimeters reliably perform this task. However, altitude was selected as a certified truth source target to demonstrate our GD-MAGS technique used in this ML semi-automated selection approach towards an eventual MLOps implementation. This aspect of the case study is an example of the meta-algorithmics application within YAMM.

This case study developed 12 combinations from a DL algorithm, 3 data preprocessing algorithms, and 2 different DL training techniques. We refer to these 12 combinations as our 12 DL-based techniques and apply them to sensor fusion for the target data set. These 12 DL-based techniques are derived from a combination of other techniques to common problems seen in this data set and mentioned in Pires et al. [131]. These techniques were created from a baseline RNN algorithm variant called Long Short-Term Memory (LSTM) and 11 other combinations of data ingest/preprocessing and batch mode techniques. We explore the advantages of applying a supplemental data modeling technique which ingests additional data for situations where there is insufficient data to properly train RNN models. Since data filtering techniques are common prior to training models with empirical data, we apply two such techniques on three noisy signals providing unexpected and conflicting values. We also apply a classification technique since data classification can lead to more accurate regressive DL models. Additionally, two batch mode techniques are implemented to evaluate the differences between the two.

For developing and deploying sensor fusion solutions, engineers need a resource efficient and systematic approach to select the most accurate DL technique with optimal hyperparameter settings. Training and testing a combination of DL techniques and LSTM hyperparameters on a statistically significant and representative subset of the data can reduce time and resource usage. From here, it follows that the selected model will likely perform well when trained and tested on the full data set.

Section 4.2 describes the SysML model of this example data sensor system and the specification of the system required in the model to be able to perform a system analysis of each DL-based technique once the best models are selected from the GD-MAGS approach.

Section 4.3 describes the example data set used for this case study that represents data collected from operating the aircraft data system. The collection of and the application of meta-algorithmic's principles to operational data is the cornerstone of YAMM. Therefore, this data has been chosen specifically because it is raw recorded data from operational commercial aircraft.

Section 4.4.1 describes the GD-MAGS approach, the 12 DL-based techniques developed for this case study, and then implements GD-MAGS to select the best DL-based techniques trained on the example data. Section 4.5.1 presents the results of this selection process.

Section 4.4.2 describes the usage of our ML model-cards with the addition of algorithm performance and resource usage to inform the system analysis which includes an analysis of alternatives and Monte Carlo analysis. Section 4.5.2 presents the results of this system analysis.

Section 4.6 discusses the results and provides the conclusions of the application of YAMM to the aircraft sensor data system case study.

4.2 Aircraft Data System Model

According to the YAMM lifecycle steps for legacy systems defined in Chapter 3 we started with developing a data system model limited to the portions applicable to the study or new capabilities. These new capabilities in this case study represent a new sensor fusion algorithm. The work in Section 4.4.1 to analyze and create algorithms from operational data can be done in parallel with this modeling step.

We developed a representative aircraft sensor data system SysML model to represent a sensor architecture for the NASA data set. We determined that this system model did not need the full simulation-enabled capabilities in order to meet our case study objectives based on the automation of 'closing the operational data feedback loop'. A SysML Internal Block Diagram (IBD) of this system is shown in Figure 4.3. This system contains two serial buses (BUS1 and BUS2), five sources of sensor and flight data, one pilot display, and one mission computer. The mission computer contains the sensor fusion algorithm software and the pilot display shows the pilot the predicted altitude from the sensor fusion model. Figure 4.3 also labels the data flowing between the computers, sensors and display across the two buses, that are limited throughput resources. This SysML data system model also defines the composite data signals flowing across these buses and these data signal definitions are shown in Figure 4.4. Figure 4.4a shows a SysML Block Definition Diagram (BDD) of the decomposition of the BUS1 and BUS2 sensor data. Figure 4.4b

shows a BDD of the data further decomposed into individual data elements from the NASA data set. The data elements are defined by signal frequency and element size in bits. The data information was extracted from this SysML system model to build the SysML Model Card Yet Another Markup Language (YAML) file shown in Appendix B. This YAML file contains the SysML Model information shown in Figure 4.8 and is used to calculate the RNN Model Card's throughput cost.

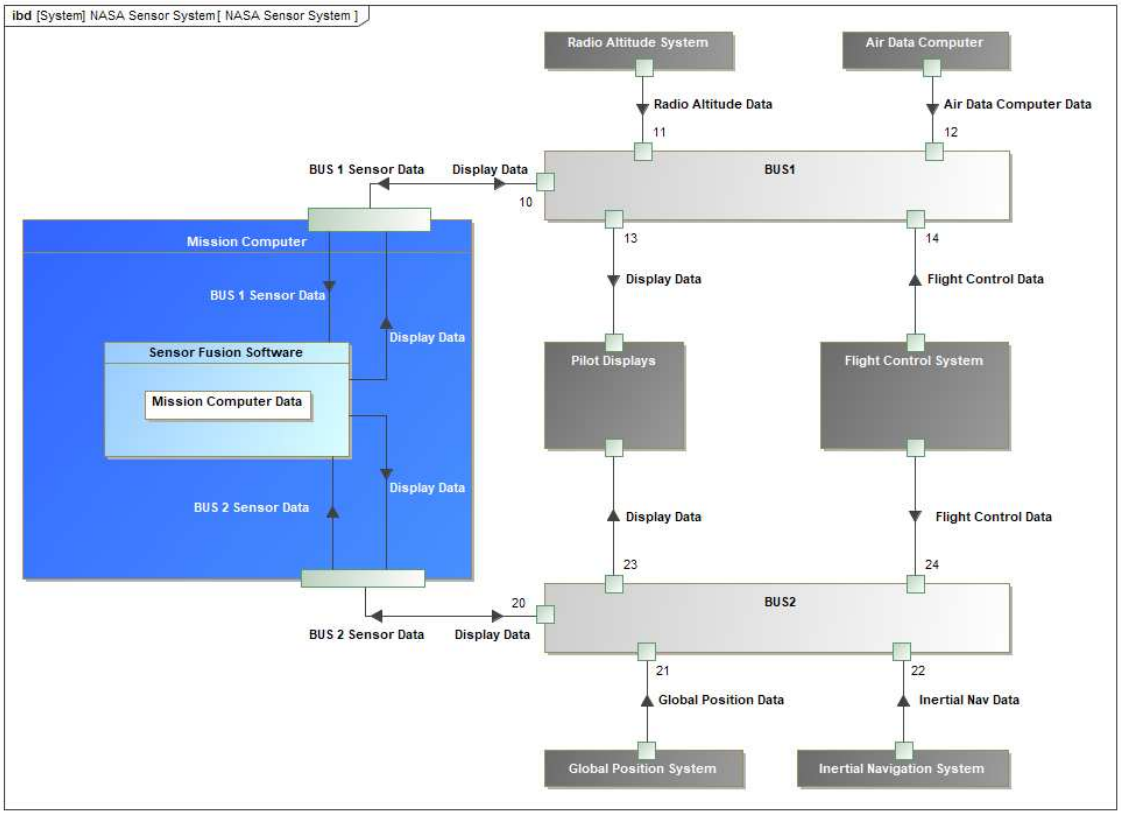
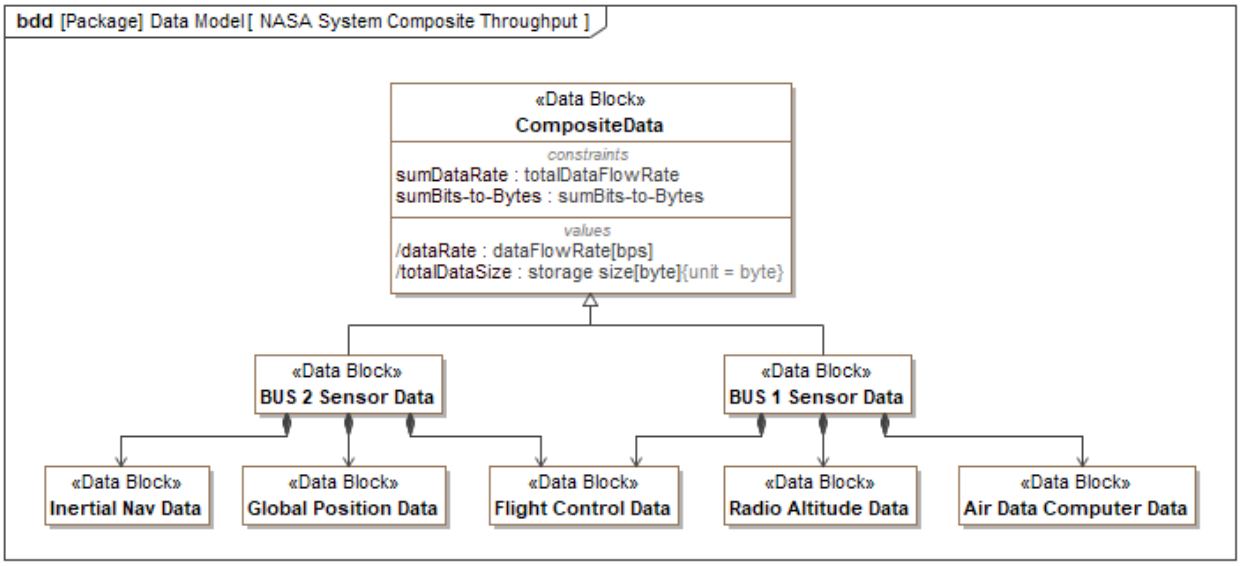


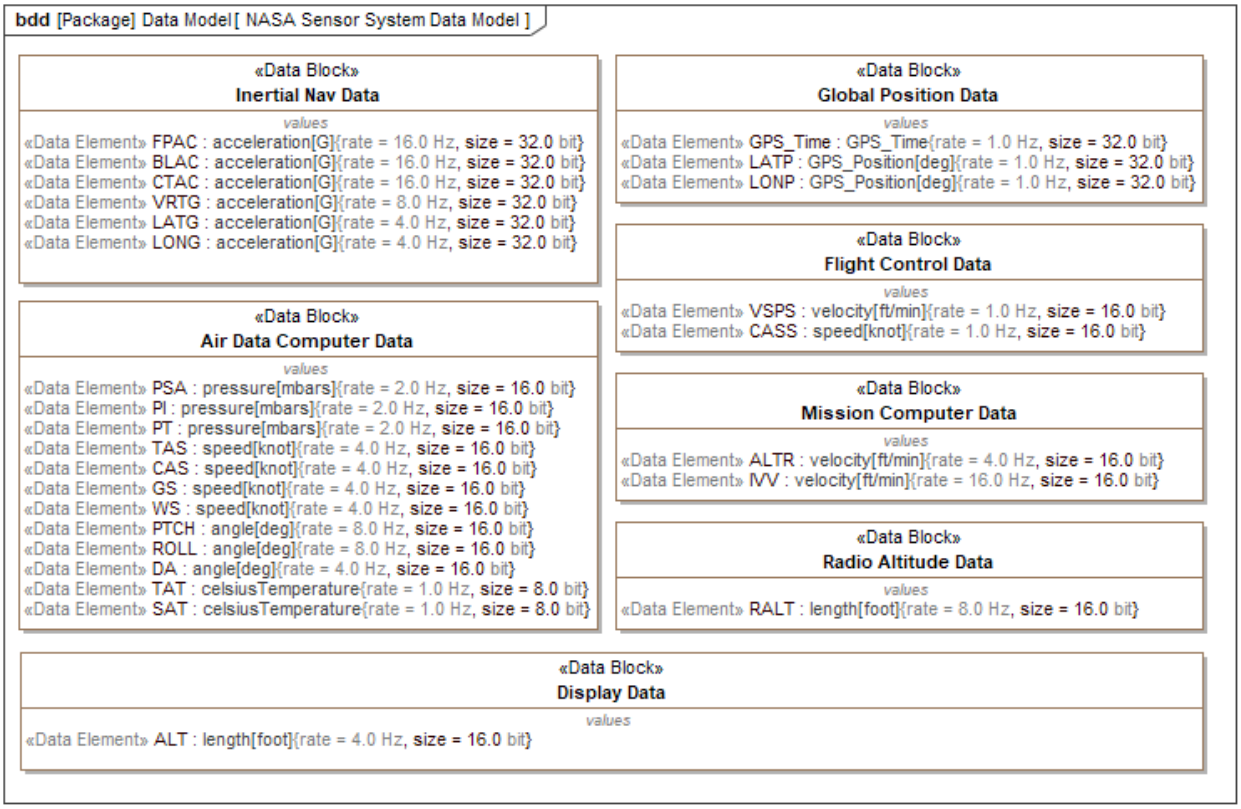
Figure 4.3: SysML IBD of the sensor data system

4.3 Aircraft Ops Data Collection

This section describes the NASA data set [132] utilized to demonstrate YAMM on the aircraft sensor data system case study. The data was recorded onboard multiple aircraft with the same sensor suite of a single type of regional jet operating in commercial service. The data files contain



(a) Composite data



(b) Data elements

Figure 4.4: SysML BDD of the data model

detailed aircraft dynamics, system performance and other engineering parameters captured on the data recorder.

The data set contains information for 35 different aircraft tail numbers averaging 5,355 flight files per aircraft with an average file size of 2MB. Each flight file contains an average of 125,000 sample points and represents a full flight that includes aircraft ground operations, take-off, flight, landing, and post landing ground operations. Additionally, each file has 188 aircraft time-series variables that include sensor data and other serial bus messages. These messages varied in recorded frequency between 0.25 and 16 Hz. Using prior aircraft system specific knowledge, we manually identified 26 variables as DL model inputs with an altitude target variable for a total of 27 variables. Of these 26 input variables there are 8 different data modalities. The sensors on this aircraft collected heterogeneous data such as altitude, altitude rate, acceleration, pressure, temperature, speed, orientation, and position. While several chosen variables, such as Selected Airspeed, Selected Vertical Speed, and Ground Speed may not be required for the DL approaches to accurately model altitude, we included them to mimic some superfluous and possibly conflicting data.

Table 4.1 shows the 26 input variables and the target variable (`ALT`) and also includes units, recorded frequencies, and brief descriptions provided by the NASA source files. We created continuous data at 16 Hz from signals recorded at lower rates using the pad method from Python `pandas` as a requirement for input into the LSTM algorithm and part of the data preprocessing steps. After this case study was originally published [26] we learned about a more efficient method by augmenting `Trino` with `DuckDB`'s AsOf Join and `PyTorch` [133] to solve the issue with asynchronous data.

Figure 4.5 shows a single flight within our data set with 11 of the 26 variables from Table 4.1 aligned and plotted against `Time` to provide insight into a few of the challenges with fusing multimodal sensor data. For instance Figure 4.5(a) shows the aircraft's ground operations as a flat line of `ALT` before take-off and after landing. However, Figures 4.5(b) and 4.5(d) show large fluctuations in altitude rate (`ALTR`) and body long acceleration (`BLAC`), respectively, while on the ground which is in conflict with `ALT`. Additionally, Figure 4.5(c) shows unrealistic periodic data supplied from vertical, lateral, and longitudinal accelerations (`VRTG, LATG, LONG`). The top and bottom red colored data in this subplot both belong to `VRTG` data. Realistic `VRTG` values for passenger aircraft

Table 4.1: Sensor variable names and descriptions

Name	Units	Rate(Hz)	Description
ALT	feet	4	Pressure altitude LSP **(Truth Source)**
Time	secs	16	Time in GMT seconds
FPAC	G	16	Flight path acceleration
BLAC	G	16	Body longitudinal acceleration
CTAC	G	16	Cross track acceleration
VRTG	G	8	Vertical acceleration
LATG	G	4	Lateral acceleration
LONG	G	4	Longitudinal acceleration
RALT	feet	8	Radio altitude LSP
ALTR	ft/min	4	Altitude rate
IVV	ft/min	16	Inertial vertical speed
VSPS	ft/min	1	Selected vertical speed
PSA	mbars	2	Average static pressure
PI	mbars	2	Impact (dynamic) pressure
PT	mbars	2	Total pressure
TAS	knots	4	True airspeed
CAS	knots	4	Calculated airspeed
GS	knots	4	Ground speed
WS	knots	4	Wind speed
CASS	knots	1	Selected airspeed
PTCH	deg	8	Aircraft pitch angle
ROLL	deg	8	Aircraft roll angle
DA	deg	4	Aircraft drift angle
TAT	°C	1	Total air temp
SAT	°C	1	Static air temp
LATP	deg	1	Latitude position LSP
LONP	deg	1	Longitude position LSP

are centered around 1 G so the periodic data shown near -3 G's are considered noise. The `LATG` and `LONG` data in this same subplot overlap significantly and have realistic values for passenger aircraft centered around 0 G's. However, they both exhibit similarly noisy behavior with unrealistic and periodic data at -1 G. These challenges in outliers and spurious data, conflicting data, and data imperfection mentioned above as well as data modality operational timing, and static vs dynamic phenomenon drove the need to develop the 12 different DL approaches described in Section 4.4.1 that are part of the GD-MAGS selection process.

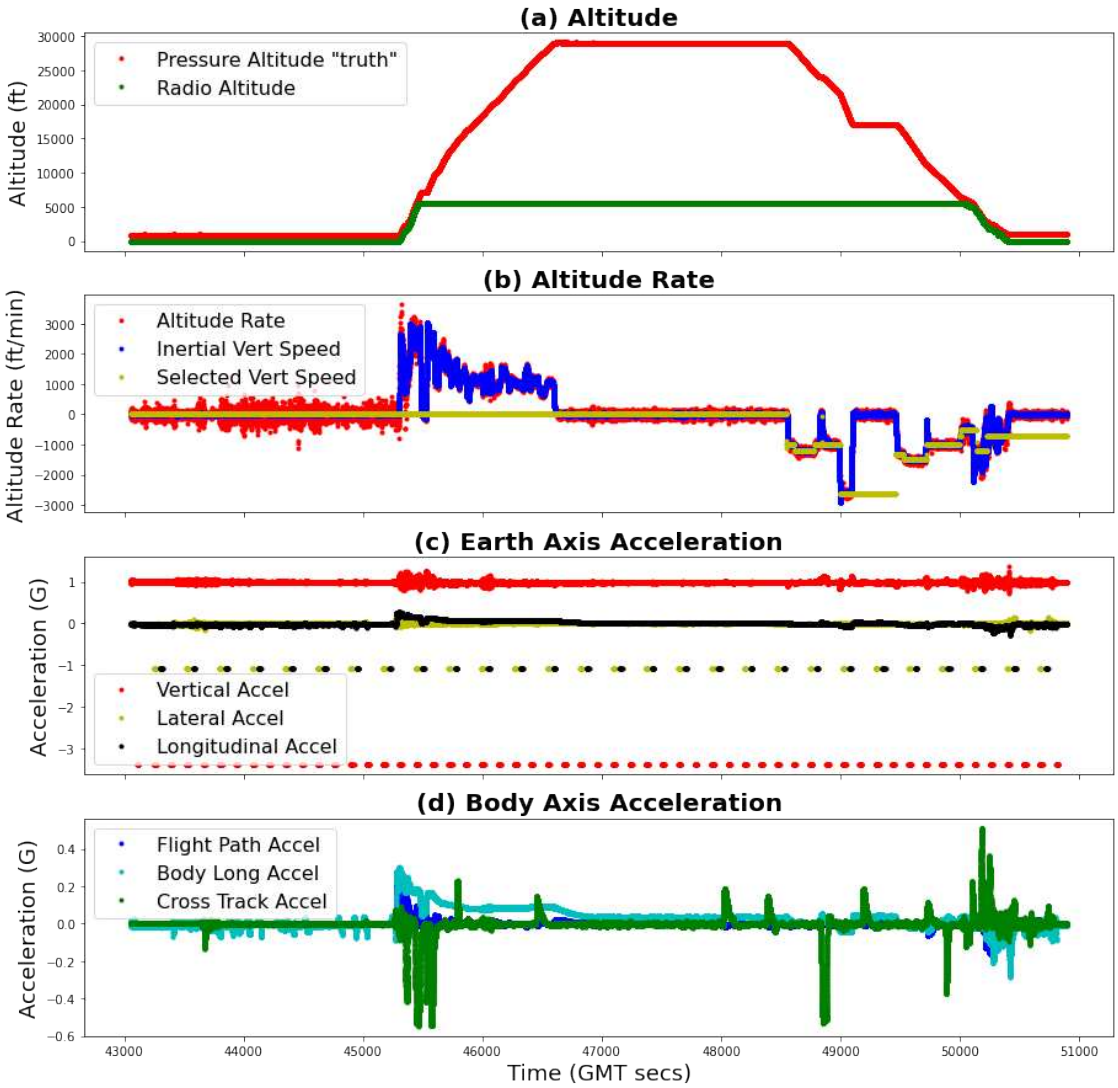


Figure 4.5: Aircraft altitude, altitude rate and acceleration

4.4 Aircraft Meta-Algorithmics Approach

In terms of the YAMM life cycle flow for legacy data systems, the ML selection approach described in Section 4.4.1 can be carried out in parallel with the development of the data system model. However, the ML analysis approach is dependent on results from the ML selection process and the system modeling process. Section 4.4.2 describes the ML analysis process that comes after the ML selection process and how it integrates ML model development into system analysis and design.

4.4.1 ML Selection Approach

Figure 4.6 is a SysML IBD of the GD-MAGS optimization architecture as it is applied to this aircraft sensor data system and sensor fusion case study. The `DL_Training` application shown creates the sensor fusion model using the `LSTM` algorithm with the `Adam` optimizer from the `PyTorch.NN` module. `DL_Training` trains each NN model using input training data collected from system operations, a DL technique, and hyperparameters provided by the `Bootstrap` application. `DL_Training` measures the model accuracy by comparing its output to the target altitude in the testing data using `PyTorch MSELoss` to calculate the performance measure, MSE. The MSE function is based on (4.1), where n is the number of data points, and T_t and Y_t are the target and predicted values at time (t).

$$MSE = \frac{1}{n}\sum_{t=1}^{n}(T_t - Y_t)^2 \tag{4.1}$$

The prune and tune loop shown with green item flows in Figure 4.6 contains the *Prune_N_Tune* procedure which includes the GD-MAGS optimization rules. The methodology starts with an initial DL Configuration matrix that contains the DL techniques and initial hyperparameters. *Prune_N_Tune* starts the first iteration by sending this initial DL configuration matrix to `Bootstrap` which completes 3 iterations of the bootstrap loop, shown with blue items flows, to calculate an average MSE. During each iteration, `Bootstrap` provides `DL_Training` 13 random training files, 7 random test files and a single DL configuration to produce a single model and MSE measurement. `Boostrap` runs all configurations in the DL configuration matrix in parallel, after which, it returns the average model MSE for each configuration to *Prune_N_Tune*. Calculating the average MSE reduces stochastic data from affecting the selection results. Before starting a new refinement iteration, the minimum MSE of the DL configuration matrix is checked against the user specified convergence criterion. The refinement iterations will continue until this convergence criterion has been met or the optimal solution has been found.

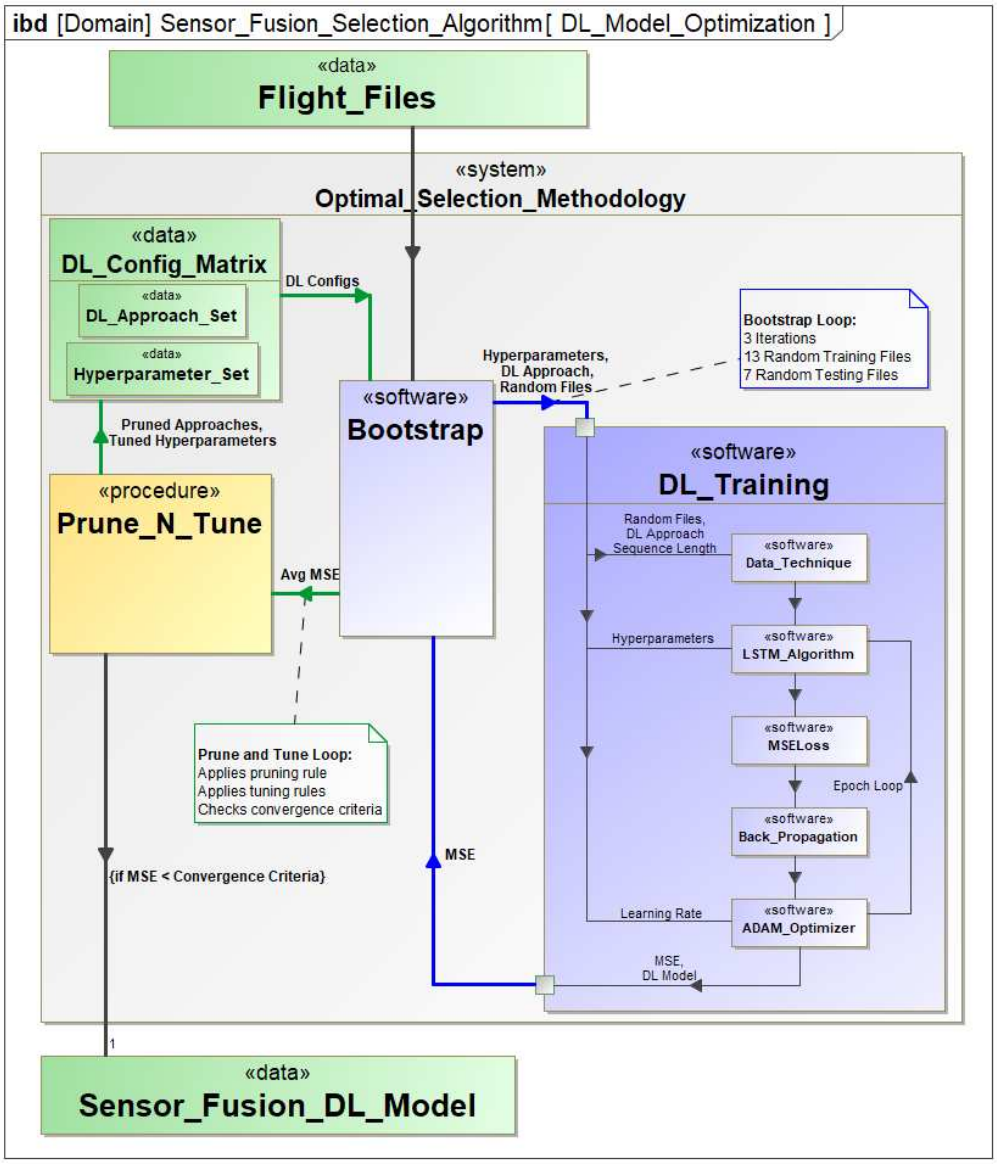
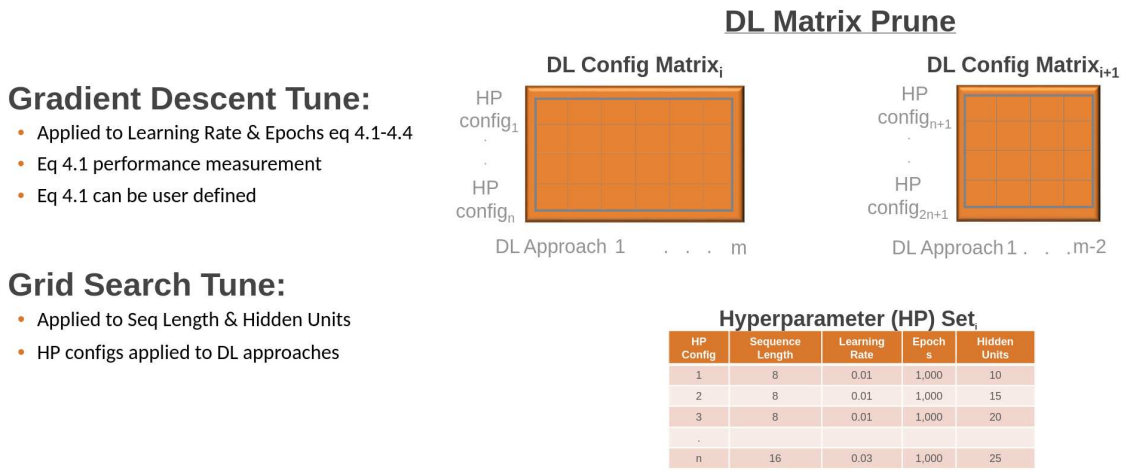


Figure 4.6: SysML diagram of Optimal Selection Methodology for sensor fusion.

The criterion can be an absolute MSE value, a gradient/optimality condition, or one of the user's choosing. When the criterion is satisfied, `Prune_N_Tune` will stop the refinement iterations and export the selected best performing model embedded with a Model Card containing performance metrics and all information needed to reproduce it. ML Model Cards will be discussed further in Section 4.4.2 describing the ML system analysis approach. If the convergence criterion isn't satisfied, a new refinement iteration begins and *Prune_N_Tune* applies the pruning and hyperparameter tuning rules based on the average MSE gradient objective function. *Prune_N_Tune* creates a new

DL configuration matrix comprised of the pruned set of techniques and the newly tuned set of hyperparameters and sends this to `Bootstrap` which repeats the refinement iterations within the selection methodology until convergence is reached. Figure 4.7 visually shows a summary of the *Prune_N_Tune* procedure, equations, and the composition of the DL configuration matrices as the prune procedure removes DL approaches.



HP Config	Sequence Length	Learning Rate	Epochs	Hidden Units
1	8	0.01	1,000	10
2	8	0.01	1,000	15
3	8	0.01	1,000	20
.				
n	16	0.03	1,000	25

Figure 4.7: *Prune_N_Tune* Process Equations and Matrices

4.4.1.1 DL Algorithm and Preprocessing Techniques

This section describes how each DL technique was created from a baseline LSTM technique for the NASA data set. The GD-MAGS methodology performs best on a large set of DL technique combinations to better identify the optimal model for the data set. To build the other 11 DL techniques from the baseline LSTM technique several procedures were applied which include: two data filter procedures, one supplemental data model procuedure, one classification procedure, and two batch mode procedures. Table 4.2 lists the names of these techniques that are compiled from the combination of procedures used to build each one. Also listed are the number of sensor variable inputs from the NASA data set used for the LSTM algorithm.

In Table 4.2, the *Baseline* or *Base* prefix refers to the baseline DL-based technique that implements the LSTM algorithm without additional procedures. This technique sets a frame of reference

Table 4.2: Deep Learning Techniques

#	Technique Name	Input #
1	Baseline	26
2	Base+Outlier	26
3	Base+RemoveVar	23
4	Base+Outlier+Model	27
5	Base+RemoveVar+Model	24
6	Base+Class	26
7	Base+Class+Outlier	26
8	Base+Class+RemoveVar	23
9	Base+Class+Outlier+Model	27
10	Base+Class+RemoveVar+Model	24
11	Base+Flightbatch	26
12	Base+Outlier+Flightbatch	26

for comparison with the 11 other techniques that build on this baseline with other preprocessing steps after the meta-algorithmic fashion of combining multiple algorithms. The following paragraphs describe the implementation of the six procedures used. These procedures are applied to the data in each technique in the order they appear in the technique names in Table 4.2 and are applied to the data before it is fed to the LSTM algorithm. The *Flightbatch* technique is an exception and is applied during the LSTM algorithm.

The *Outlier* algorithm is the first of two data filter procedures applied in this approach. It uses the Python Scikit-Learn `IsolationForest` outlier algorithm chosen for its performance against other well known outlier detection algorithms for data sets larger than 1,000 samples [134]. The `IsolationForest` algorithm was used to evaluate its effect on the model's accuracy by reducing outliers, or *noise*, in the three noisy signals discussed in Section 4.3 when introducing the NASA data set.

RemoveVar is the second data filter procedure applied which removes the three noisy variables (`VRTG, LATG, LONG`) from the input, thereby bypassing the need for outlier detection all together. The goal is to compare its model performance against the Isolation Forest algorithm performance.

The *Model* algorithm addresses possible limitations of the aircraft's sensors and available data recorded. This is a supplemental data model that provides ground elevation data as input to the LSTM algorithm. The intent is to provide additional data to the LSTM algorithm to more accurately model the target altitude (`ALT`) through a Digital Terrain Elevation Data (DTED) table lookup from the GMTED2010 [135] database. This database is commonly used on aircraft or can be added to the aircraft's software if needed.

The *Class* algorithm classifies the flight data as either ground operations or in-flight data to remove conflicting ground data from the training set. This procedure deletes all ground data before the LSTM algorithm is trained on the rest of the sensor data, towards improving the final solution.

Two different batch mode training algorithms are included to analyze the effects of splitting time series data in two ways when training models. The first batch method, which is implemented in the baseline technique, appends the flight data from all 13 training flight files into a single data frame and trains the model in equal batch sizes of 50,000 data points at a time. This training algorithm ignores the beginning and end of individual flight files and iteratively trains forward through the model, calculates loss, back-propagates the loss, and performs the single optimization step for each batch file until moving onto the next epoch. The second batch mode training algorithm, known as *Flightbatch*, customizes each batch size to contain each of the 13 flight files and performs the same back-propagation steps as the baseline which is also describe in Yao et al. [90].

For each technique, the algorithms above are applied to the input data, then the data is scaled near unity using scikit-learn `MinMaxScaler` and `fit_transform` algorithms to reduce truncation and rounding errors. Next, a sequence length sized windowing scheme is applied. Finally, the training and testing data partitions are created and the model is trained. The testing partition is scaled with the training `MinMaxScalar` values with scikit-learn's `transform` function before measuring MSE.

4.4.1.2 Initial LSTM hyperparameters

Initial learning rates (LR) and numbers of epochs (Epochs) are chosen for quick model convergence, then tuned in subsequent iterations. The purpose is to eliminate relatively poor performing

techniques early in the process when it is computationally cheap. Initial values for number of hidden units (Units) were spread out evenly to produce an initial MSE gradient to inform the tuning rules. Sequence lengths 8 and 16 were chosen as factors of the 16 Hz signals. Selection of the RNN hyperparameters, guidance for initial values, and why we chose the Adam optimization algorithm are summarized here for completeness, however a more thorough explanation of heuristic techniques for selecting these settings can be found in the book Deep Learning [87]. For this procedure we used trial and error on a few training sets to figure out a good set of initial hyperparameter values that showed quick convergence across all 12 DL-techniques. The Adam optimizer is a method for stochastic optimization and is used frequently in ML algorithms and was chosen for its general stability and performance. Unfortunately, there is not a deterministic method for setting these values, so a heuristic approach is needed.

4.4.1.3 GD-MAGS Optimization Rules

This section describes the DL technique *pruning* rule and the gradient descent hyperparameter *tuning* rules applied to the grid search method across concurrent DL techniques.

Technique pruning rule: For each GD-MAGS refinement iteration, remove the two worst performing DL techniques from the DL configuration matrix. Here, performance is based on the lowest MSE of best hyperparameters for the technique. More than two techniques can be removed per iteration if there is a consistent trend of poor technique performance across iterations. We recommend caution when removing more than two techniques per iteration before the third iteration, since it is difficult to spot a positive or negative trend using only two points of reference.

Learning rate tuning rule: The MSE ratio of the best and worst performing hyperparameter configurations is the basis for the learning rate and epoch number tuning rules. The learning rate is reduced proportionally to this gradient with a diminishing damping factor applied. The factor slows the learning rate reduction early in the process when the gradient is large. However, this factor is reduced at each iteration as the solutions converge. The new learning rate for the next

iteration (LR_{i+1}) is calculated using (4.2),

$$LR_{i+1} = \frac{LR_{i_{best}}}{ratio_{LR}} \tag{4.2}$$

where $LR_{i_{best}}$ is the best performing hyperparameter configuration learning rate for the current iteration i. The $ratio_{LR}$ is the learning rate ratio calculated in (4.3),

$$ratio_{LR} = \frac{1}{F_d}\frac{MSE_{worst}}{MSE_{best}} \tag{4.3}$$

where F_d is the diminishing damping factor, MSE_{worst} and MSE_{best} are the worst and best performing hyperparameter configuration's average MSE, respectively. The ratio $\frac{F_d MSE_{worst}}{MSE_{best}}$ must always remain greater than one and initial recommended values of F_d should produce LR_{i+1} less than 1 order of magnitude (OOM) smaller than LR_i. Very large initial F_d will drop the LR_{i+1} several OOM leading to convergence that will be overly resource intensive and the methodology will most likely skip optimum solutions. Very small F_D will increase LR_{i+1} compared to LR_i and lead to divergent solutions.

Epoch number tuning rule: Assuming the previous models were sufficiently converged, the number of epochs for the next iteration ($Epochs_{i+1}$) are adjusted using (4.4),

$$Epochs_{i+1} = Epochs_{i_{best}}\left(\frac{LR_{i_{best}}}{LR_{i+1}})\right) \tag{4.4}$$

where $Epochs_{i_{best}}$ is the Epoch number of the best performing hyperparameter configuration. If there is indication this hyperparameter configuration sufficiently converged at a lower Epoch number, then reduce $Epochs_{i_{best}}$ to the Epoch number at convergence before calculating $Epochs_{i+1}$. These adjustments to Epoch numbers are meant to prevent computational resource waste for too many Epochs, or insufficient convergence for too few.

Hidden unit tuning rule: Hidden unit tuning can be difficult to implement and performance of the hidden units can change between iterations. Therefore, it is recommended to keep a range

of hidden units throughout hyperparameter tuning, as suggested by the grid search procedure, to provide additional coverage to find the optimal solution.

Sequence length tuning rule: The sequence lengths are also adjusted once performance trends are observed. Consideration of the data set frequency and data ingest method is important when applying this tuning rule. For instance, this data set's primary recording rates are 16, 8, and 4 Hz, therefore recommended sequence lengths are equal to the frequencies found in the data. Otherwise, stale data created by the pad method for the 8 and 4 Hz signals can reduce overall performance of the model other sequence lengths are used.

4.4.1.4 GD-MAGS Algorithm Verification

The accuracy of our ML selection approach is verified by using 20 random flights each from two other randomly chosen aircraft tail numbers within this data set. The verification executes the bootstrap loop in Figure 4.6 using the random data from the other aircraft and three pre-trained models of varying performance from the selection process. The optimally selected pre-trained model and two other lower performing models are compared for relative performance based on average MSE. This verifies that the initially trained and selected model continues to outperform non-optimal pre-trained models on other aircraft with the same sensors.

Once the top ML models are selected by the GD-MAGS algorithm the next step in our ML selection and analysis approach is to analyze these ML models for their resource usage on the system in which they will be executed on.

4.4.2 ML Trade Study Analysis

This section represents the *MDAO* stage of the YAMM life cycle as shown in Figure 4.2. The focus is on the aircraft sensor fusion example as a continuation of Sections 4.2 and 4.4.1, where the system model was built and the meta-algorithms were applied that developed and selected the best DL models, respectively.

Figure 4.8 shows the sensor fusion system analysis process being evaluated in this approach. For clarity, this is the SysML process model and is separate from the sensor system SysML model

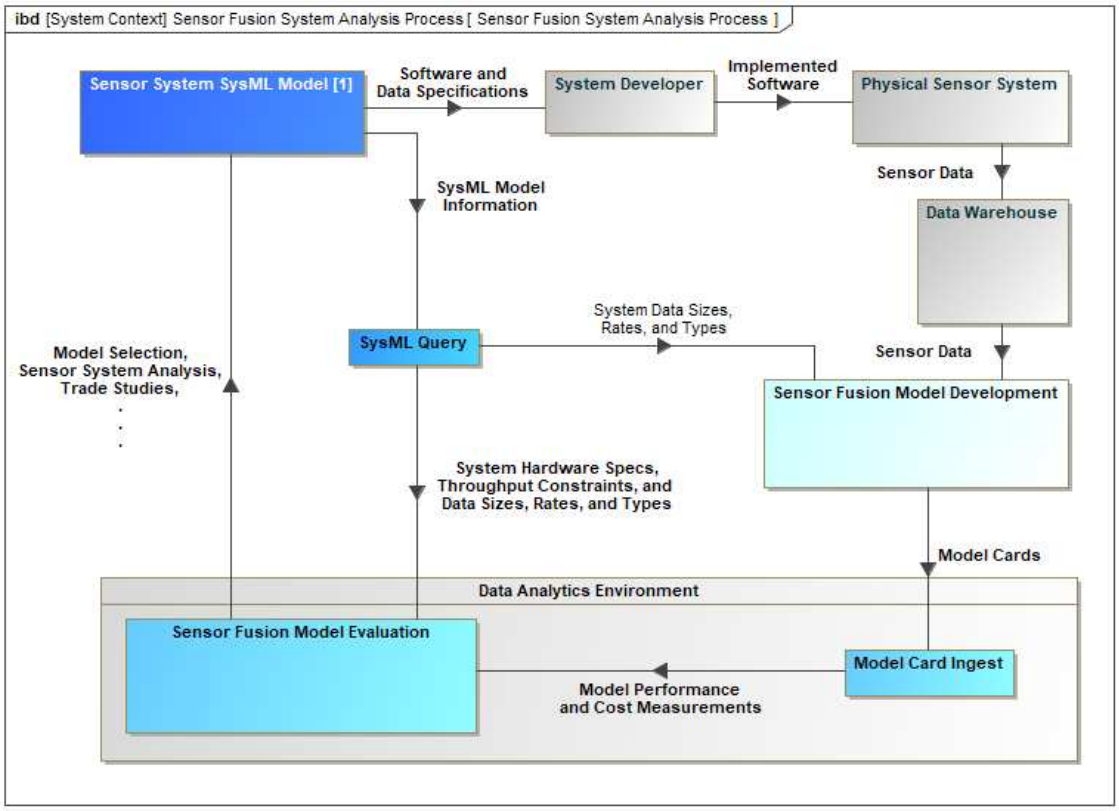


Figure 4.8: SysML IBD of the system analysis process

(i.e. SysML system model). This figure shows a *SysML Query* that reads *SysML Model Information* from the *Sensor System SysML Model* and writes the system specifications and data to a SysML system Model Card YAML file (Appendix B) that will be read by the *Sensor Fusion Model Evaluation* and *Sensor Fusion Model Development* applications. The *Sensor Fusion Model Development* application represents the model selection process of the previous study in Section 4.4.1 that generated the RNN models used in this ML analysis process. For this case study, the Sensor Fusion Model Development reads the SysML Model Card, loads the previously trained RNN models, evaluates the models on a benchmark PC, and creates an RNN Model Card (Appendix A) for each of the models. After the RNN Model Cards are created, the *Model Card Ingest* application reads each RNN Model Card and provides the information to the *Sensor Fusion Model Evaluation* analysis application. This application executes an AoA application based on the system objective function, then executes a Monte Carlo analysis application to return the distribution of possible

objective function values given a range of random objective variable values. The results of these analyses provide information back to the system engineers who use it to modify the SysML system model specification, which informs the next iteration of sensor fusion software development.

In a production environment the SysML system model would also provide hardware, software and data specifications to a team of system developers that would build or update the physical sensor system as described in section 4.2. Once the physical sensor system is built, the data it generates is processed and stored in yet another data system. The hybrid cloud case study is an example of one of these data systems used to store sensor and aircraft data for the *Sensor Fusion Model Development* users to utilize to iterate on their sensor fusion solution.

Table 4.3 shows the top four techniques selected from Section 4.4.1 for application of our ML analysis process. This table also shows the RNN hyperparameters for each DL model that could also contribute to additional resource costs on the aircraft system. For instance, a sequence length of 16 requires 16 of the previous time steps to be stored and used to predict the altitude at the next time step. This could increase the volatile memory resources needed and possibly affect the inference latency. Also, larger RNN hidden units could possibly increase the volatile memory used during model predictions. Additionally, more DL model inputs could directly increase the amount of throughput resources consumed by the DL model. However, throughput is a function of variable sizes and update rates, so the impact to aircraft system resources depend on variable size and rate as well.

Table 4.3: Top 4 models from the ML selection process

DL Technique	**N Inputs**	**Seq. Length**	**Hidden Units**
Base+RemoveVar	23	16	20
Baseline	26	8	25
Base+Outlier	26	4	25
Base+RemoveVar+Model	24	8	15

4.4.2.1 System Analysis

Two simple, but representative, system analysis processes were created and executed on this case study to evaluate the effectiveness of our ML system analysis process. The first analysis is an AoA developed for this case study which presents five different alternative scenarios for an aircraft sensor system. The second is a Monte Carlo analysis that looks at a random distribution of 10,000 different scenarios to see the range of objective function performance for each model on the system.

We created the objective function defined in Eq. (4.5) for this case study to score each model's system performance and cost. Lower values of this objective function indicate a better score. MSE_B is the MSE of the model on the benchmark data, MSE_G is a measure of the model's ability to generalize well and is calculated by Eq. (4.6), where MSE_{avg} is the bagging MSE average during the bootstrap testing phase. $Infer$ is the model inference latency, P is the data pre-processing latency, T is the system data throughput required for the sensor fusion model, $vMem$ is the volatile memory consumed during inference, $nvMem$ is the non-volatile memory needed to store the static model. Each of these quantities are multiplied by coefficients C_1 - C_7 to modify the weight each quantity has on the objective function. These seven coefficients must always add up to 1.

$$f = \sum(C_1 MSE_{Bn}, C_2 MSE_{Gn}, C_3 Infer_n, C_4 P_n, C_5 T_n, C_6 vMem_n, C_7 nvMem_n) \tag{4.5}$$

$$MSE_G = |MSE_{avg} - MSE_B| \tag{4.6}$$

All performance and cost variables were normalized in Eq. (4.5), as indicated by the subscript $_n$, using the Min-Max Scaling method shown in Eq. (4.7). Where $min(x)$ is the minimum value across each of the four models, $max(x)$ is the maximum across the four models, x is the original model value and x' is the normalized value on a scale of [0,1]. Normalization reduces number

biases of very large or very small objective function variables.

$$x' = \frac{x - min(x)}{max(x) - min(x)} \tag{4.7}$$

We developed five different scenarios for the AoA and estimated the seven weighted coefficients for each of them. Figure 4.9 shows a plot each of the seven weighted coefficients values applied to each performance and cost term in the objective function. The objective function weighted coefficients for each of the five different scenarios are plotted next to each other to show the differences. The first scenario in this AoA is one in which the aircraft has practically unlimited resources and therefore the MSE and MSE_G are weighted the most heavily, and inference and pre-processing latency are still important, but weighted less than MSE. Resource costs were weighted at almost zero. This scenario is labeled *Unlimited Resources* in the plot. The second scenario, ***MSE Meets Sys Tol***, is one in which all of the current sensor fusion models meet or exceed the overall system tolerance for accuracy. This means that additional accuracy in the models is shadowed by poor accuracy elsewhere in the system. This scenario weighs the MSE and MSE_G low while weighing inference and pre-processing speed higher since there are still timing requirements to meet. The third, fourth, and fifth scenarios are ones in which the throughput, volatile memory, and non-volatile memory, respectively, are near their constrained limits and are each weighted high for their respective scenario.

The Monte Carlo analysis for this case study operates on the same principle as the AoA. However, for each Monte Carlo run a random weight was assigned to each of the seven coefficients and they were normalized to sum to one. The objective function was calculated for this run and this was repeated for 10,000 runs to capture the distribution of performance scores possible for each model.

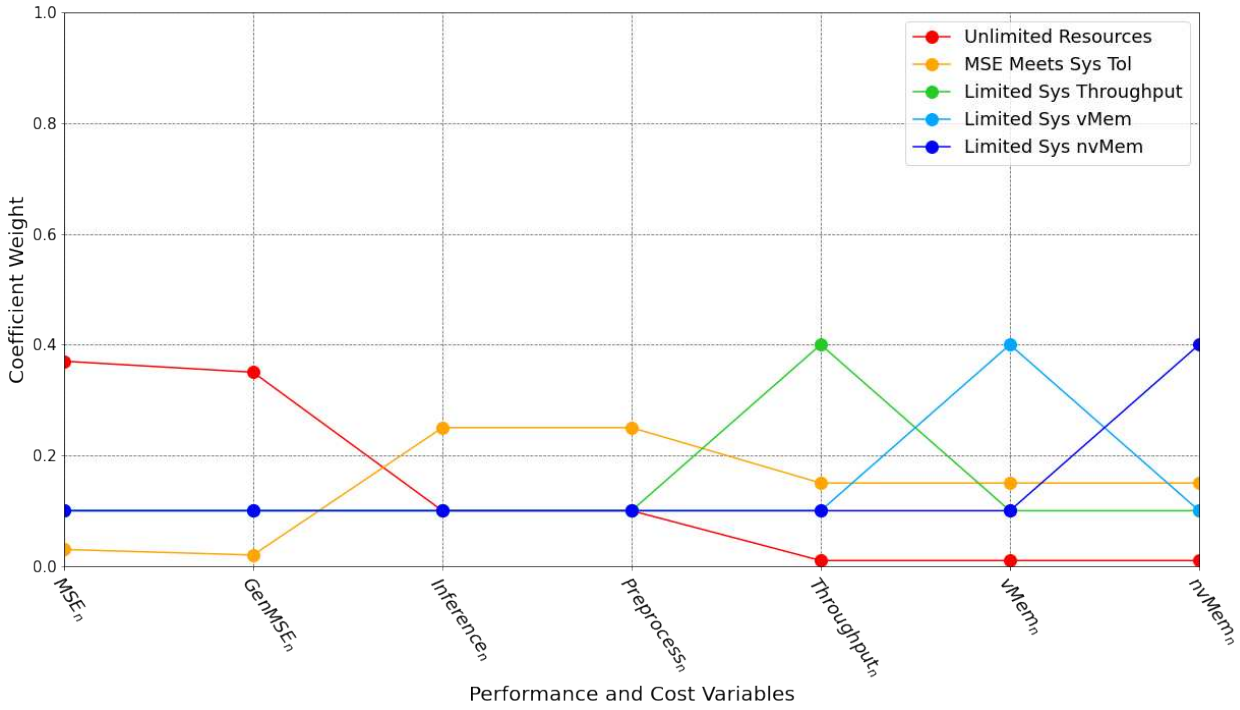


Figure 4.9: Objective function coefficient weights for AoA

4.5 Aircraft Case Study Results

This section presents and discusses the results from applying the YAMM to the legacy aircraft sensor data system towards developing a better sensor fusion algorithm. Section 4.5.1 presents the ML selection process results and Section 4.5.2 present the ML system analysis results.

4.5.1 ML Selection Results

This section evaluates the potential accuracy and flexibility of a RNN, specifically the LSTM algorithm, applied to real-time aircraft data from an open NASA data set [132]. The NASA data was recorded onboard a single type of regional jet operating in commercial service over a three year period and was chosen for its availability to publish and the known accuracy in the altitude measurement. The LSTM algorithm is trained using inputs from the aircraft multi-modal sensors with the altitude as the target variable. We evaluated the accuracy of the GD-MAGS ML selection process [26] on two other aircraft data sets of the same type with the same sensor suite to verify the selected DL model is an optimal solution for those as well. We present the ML selection results

in Section 4.5.1.1, discuss them later in Section 4.6, and summarize our conclusions about the ML selection process in Section 4.7.1.

4.5.1.1 GD-MAGS Results

Executing the GD-MAGS ML selection approach on the aircraft sensor fusion example data set (i.e. NASA data set) produced 248 unique aircraft altitude sensor fusion models before selecting an optimal solution. It started from an initial DL configuration matrix with 72 DL-techniques and hyperparameter configurations then executed the GD-MAGS approach discussed in Section 4.4.1. The initial DL configuration matrix is composed of the hyperparameters shown in Table 4.4 and all of the DL techniques previously listed in Table 4.2. Table 4.4 also lists the final selected model's hyperparameters. Model training and testing of the DL configuration matrix for each iteration was run in parallel. Section 4.5.1.2 summarizes the results from all refinement iterations and application of the rules. Section 4.5.1.3 presents the convergence of each refinement iteration's results. Lastly, section 4.5.1.4 presents the results from the methodology verification.

Table 4.4: LSTM hyperparameters

Config	SeqLen	LR	Epochs	Units
Initial Matrix				
1	8	0.01	1,000	10
2	8	0.05	1,000	20
3	16	0.01	1,000	10
4	16	0.03	1,000	20
5	16	0.05	1,000	20
6	16	0.05	1,000	25
Optimal Model (Iteration 4)				
31	16	0.0007	3,500	20

4.5.1.2 Refinement iterations

As mentioned previously in Section 4.4.1, the first step of this approach is to train and measure the performance of each DL-based technique and combination hyperparamters in the DL config-

uration matrix then use the average bootstrap loop MSE as inputs into the second iteration for pruning and tuning.

As expected, the average MSE generated from the first iteration did not meet the convergence criterion, therefore the selection approach proceeded onto iteration two. In iteration two the *Prune_N_Tune* procedure pruned DL-techniques 11 and 12 for being the worst two performing techniques. This is indicated in the $Pruned_i$ column of Table 4.5 and can be seen in Figure 4.10 by the missing data points between the iteration 1 and 2 lines.

Table 4.5: Deep Learning Technique Results

#	Technique Name	Input #	Pruned$_i$
1	Baseline	26	5
2	Base+Outlier	26	5
3	Base+RemoveVar	23	Selected
4	Base+Outlier+Model	27	4
5	Base+RemoveVar+Model	24	4
6	Base+Class	26	4
7	Base+Class+Outlier	26	3
8	Base+Class+RemoveVar	23	4
9	Base+Class+Outlier+Model	27	3
10	Base+Class+RemoveVar+Model	24	4
11	Base+Flightbatch	26	2
12	Base+Outlier+Flightbatch	26	2

The best and worst performing hyperparameter configurations, one and five in Table 4.4, respectively, were used to calculate the second iteration's learning rate and Epochs using equations (4.2), (4.3), and (4.4). An initial damping factor of 4 was chosen and is divided by 2 each successive iteration. Model convergence data indicated the number of initial epochs were larger than needed for convergence, therefore $Epochs_{i_{best}}$ was reduced before the calculations. No trends were identified within the hidden units, therefore additional values were added to expand exploration of the hyperparameters. No significant performance difference between sequence lengths were identified, therefore two additional sequence length configurations equal to 8 were added to expand coverage as well.

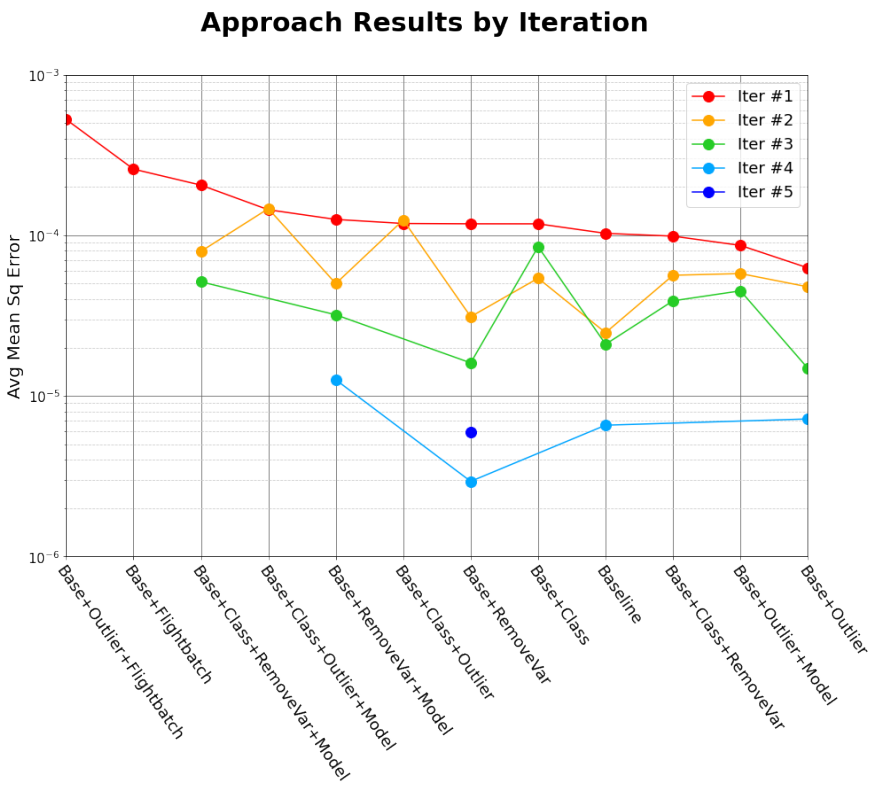


Figure 4.10: Minimum MSE Across Refinement Iterations for each DL Technique

Iterations two through four proceeded in much the same way as iteration one. The worst two DL techniques were pruned and the hyperparameters tuned according to the associated equations. However, an additional two DL techniques, techniques 4 and 8, were pruned during iteration four based on their trend of poor performance increase across iterations one through three. Iteration five proceeded with the single best performing technique. Figure 4.10 displays the progression of each iteration's best tuned techniques as the GD-MAGS process is performed on the data. This figure also indicates performance differences for DL techniques as the hyperparameters are tuned. This is observed in the inconsistent drop of MSE across DL techniques per iteration.

The results from iteration five returned a minimum MSE that was greater than iteration four's minimum MSE as seen in Figure 4.10 by the rise in MSE of the *Base+RemoveVar* technique from iteration four to five. From this observation, it was determined that the minimum had been found in iteration four and the procedural loop was terminated. Ideally, additional iterations centered around iteration four's data would further refine the optimal configuration, however finding the location of

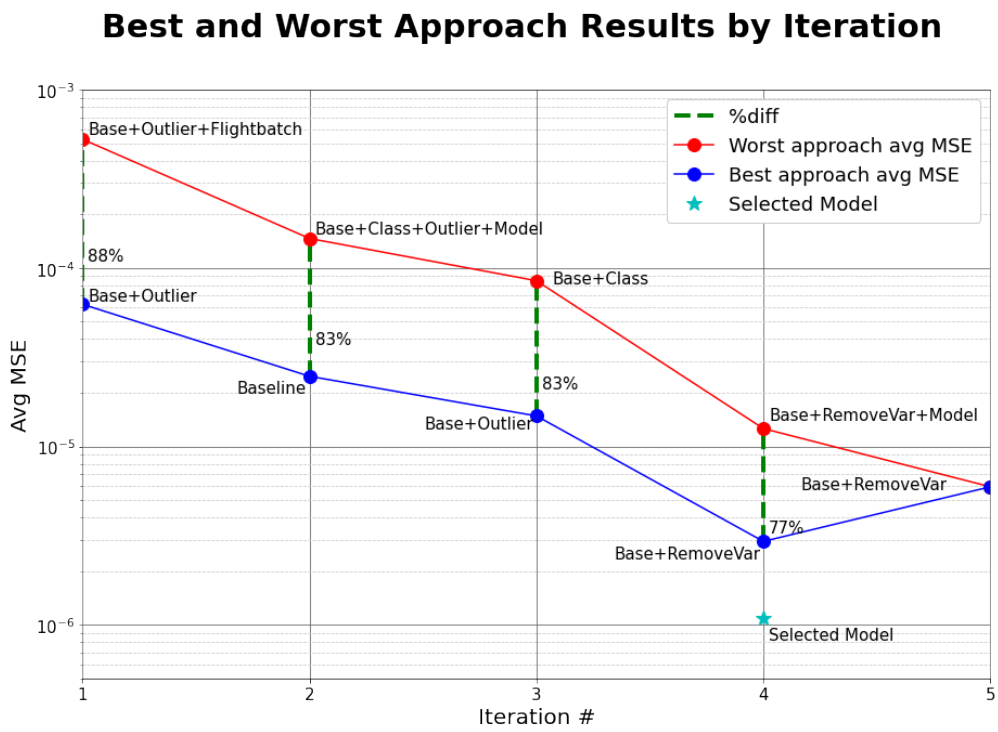


Figure 4.11: Best and Worst DL techniques across iterations

a minimum was deemed sufficient for this study. Therefore, the *Base+RemoveVar* technique and hyperparameter configuration 31 from Table 4.4 was selected as the optimal model.

4.5.1.3 Convergence results

This section presents the GD-MAGS ML selection convergence across all 12 DL techniques and the hyperparameter space. Figure 4.11 shows convergence of the best and worst DL techniques and hyperparameters as the blue and red lines draw nearer to each other. The dotted green lines and percentage labels quantify the convergence between best and worst techniques at each iteration by listing the percent difference between them. Figure 4.11 also shows the optimal bootstrap average at iteration four and the the MSE of the selected model from the best of the three bootstrap loop models. This figure also labels the best and worst performing DL techniques across the five refinement iterations. The final selected configuration in iteration four has a bootstrap loop average MSE of 2.93e-06 while the selected model has an MSE of 1.09e-06.

When comparing error of altitude output from this selected model to the seven test files used in iteration four the maximum in-flight error was 155.4 ft as the aircraft is transitioning from level flight into a descent. Larger errors are present for this comparison but they occurred in transitions between disparate flight files and aircraft post-landing maneuvers which are artifacts of the batch process and classification, respectively. These errors were not considered as in-flight errors, but were included in the standard deviation. The standard deviation of the errors across all seven flights is 29.7 ft.

4.5.1.4 Verification and Validation results

The selected ML model from the *Base+RemoveVar* technique is used to predict the altitude using the other sensor values from a random flight not previously used to train or test this model. The max error during this flight is 157ft after landing and a max in-flight error of 99ft. The standard deviation for the entire flight is 16ft. Figure 4.12(a) shows the comparison between the target, or truth, altitude from the flight's data set and the predicted altitude. It also highlights sections within the flight of the largest errors between the truth and predictions. Figure 4.12(b) shows the error between the truth and the predictions and also shows the FAA altitude tolerance [136] for altimeters as a function of altitude. The two largest errors that exceed the FAA altitude tolerances occur on the ground before take-off and after landing. The third largest error that exceeds tolerances occurs during the landing approach for this aircraft.

The selected model from the *Base+RemoveVar* technique and the two next highest performing models from iteration 4, *Baseline* and *Base+Outlier*, were selected as the models to use to validate the GD-MAGS selection of the optimal model. These models were trained on tail number 687 and the validation testing was performed on tail numbers 660 and 675 which were all chosen randomly. These three pre-trained models ingested data from the training set and two other aircraft tail numbers as validation data sets. A bootstrap loop average for each model and data set was calculated and plotted in Figure 4.13. This figure shows that the average bootstrap MSE for the optimally selected model continues to outperform the other two models on two other aircraft data sets.

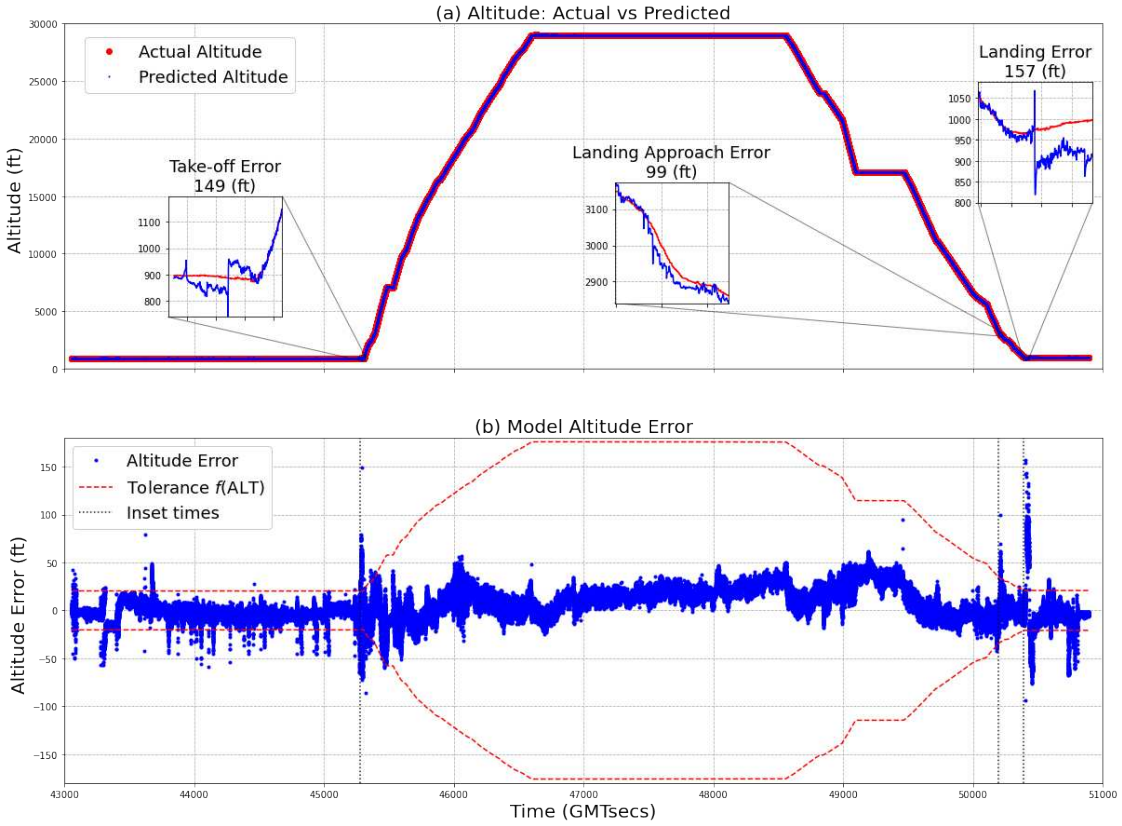


Figure 4.12: Model Predicted Altitude vs Altitude Truth and Altimeter Allowable Tolerances

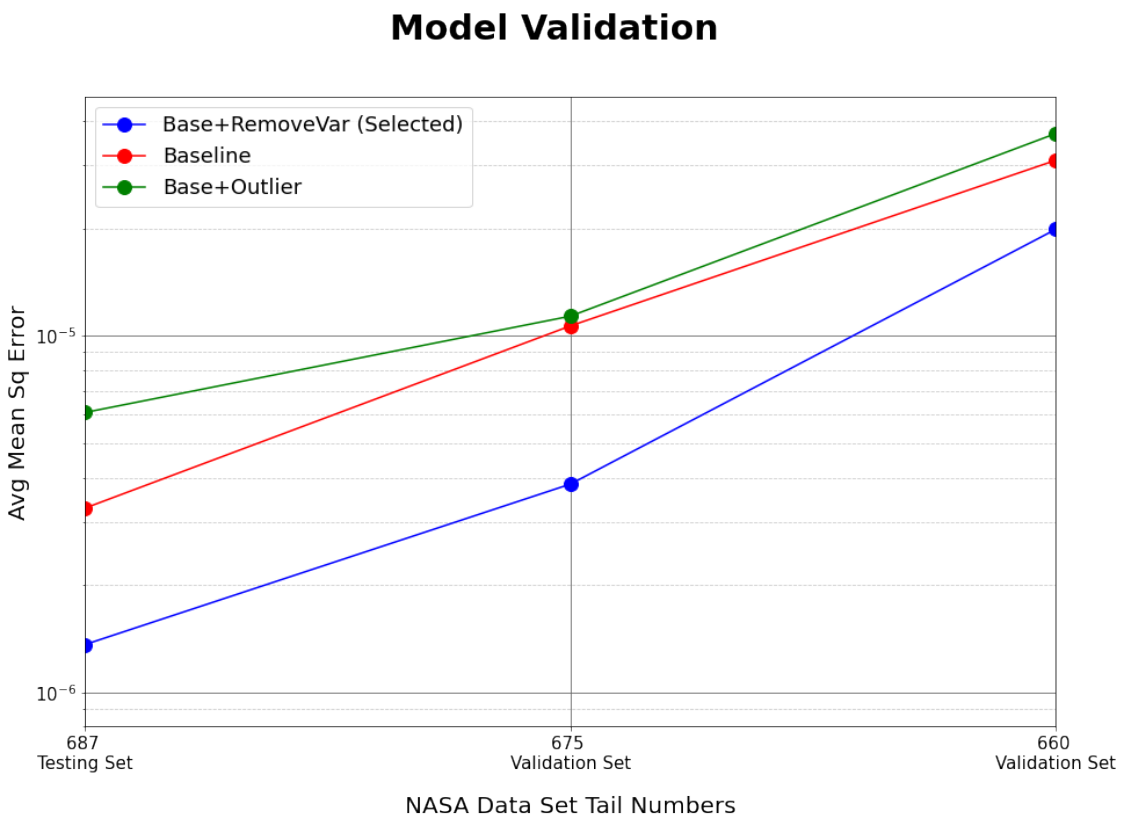


Figure 4.13: Relative Performance Verification with Two Other Aircraft Data Sets

4.5.2 ML Trade Study Analysis Results

The goal of this section is to evaluate the scalable and automated process we developed that passes ML model information to a system trade study analysis to select ML models, not only on their accuracy, but also on the resource usage to perform ML inference on their targeted systems. We accomplish this information exchange through the use of SysML system models and ML Model Cards. We refer to this as the ML analysis process and it is part of the larger effort towards DevOps-like automation of ML selection and analysis in YAMM. This trade study analysis process is executed on the aircraft sensor fusion example system to evaluate its usefulness. The goal of the case study is to choose the best sensor fusion model for a limited-resource system using pre-trained RNN-based sensor fusion models from the GD-MAGS ML selection process. An objective function scores the performance of four models as a function of network throughput, model inference latency, data pre-processing latency, accuracy, generalization, and volatile and non-volatile memory consumed. A SysML sensor system model provides system specifications, while the pre-trained DL models provide the performance and cost measurements via Model Cards. The final DL model will be chosen based on the trade study analysis using the input from the SysML model and the Model Cards to generate the output of the objective function. The novelty here is the scalable and automated process that uses the machine-readable RNN Model Cards and system input from a SysML model for system analysis and design. We then present the results in Sections 4.5.2.1 through 4.5.2.3 and discuss the results in Section 4.6.1. Lastly, we present our conclusions about the trade study automation in Section 4.7.2.

4.5.2.1 Model Cards

The ML system analysis process described in Section 4.4.2 and shown in the ML analysis process SysML diagram in Figure 4.8 was implemented and executed in *Python*. First, an example NASA data file was chosen that was not part of the training, testing or validation that happened in the original study [26] that produced the four DL techniques used in this section. In parallel, a SysML sensor system model was created with all data variables used in each of the DL techniques modeled and allocated to the appropriate sensors, computers, displays, and serial buses. This

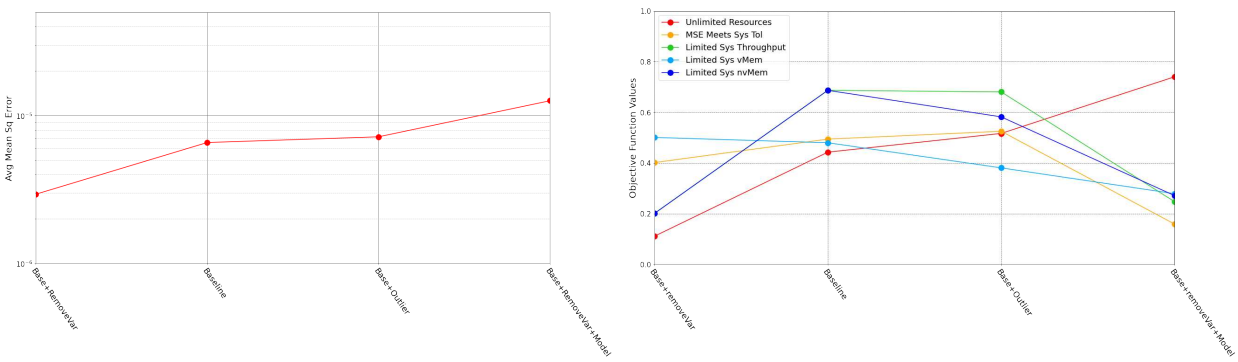
SysML data model informed the creation of the SysML Model Card yaml file. We look forward to the SysML v2 API query tool specification release to easily automate this step.

After the SysML Model Card was defined, the selected DL models were run through a series of inference test loops to measure and record the quantities needed for the RNN Model Card. We measured each inference event and repeated this 3 times to get average performance and resource usage cost measurements for the RNN Model Cards. Additionally, the performance metrics and data used to train, test and validate these DL techniques is captured and recorded in the ML model object class by the ML selection implementation [26]. This data is also documented in the RNN Model Card yaml files. Unfortunately, not all data recommended and listed in the current RNN Model Cards were captured in this previous ML selection implementation so some values are blank/*null* in the example file. The lack of data did not affect this study, but a completed Model Card is recommended for future and more complex studies.

The next two sections summarize the results from the AoA and the Monte Carlo trade study analysis using these RNN Model Cards and the SysML Model Card as inputs. These analyses are automated towards a Dev*Ops MDAO process as defined by YAMM.

4.5.2.2 Analysis of Alternatives

Once the SysML and RNN Model Cards were created they were read in and each performance and cost measurement was normalized using the Min-Max Scaling method listed in Eq. (4.7). This normalized each value on a scale of 0-1. The set of coefficients for each AoA scenario shown in Figure 4.9 and described in section 4.4.2.1 were used as inputs to the objective function (Eq. (4.5)) and the results are shown in Figure 4.14b. The objective function is a discrete minimization function, meaning lower values are better. These values represent system resource usage costs and sensor fusion error which should be minimized. Figure 4.14a shows the comparison of the error (MSE) of the top four DL techniques from the ML selection process, whereas Figure 4.14b shows the relative performance of these DL models based on the objective function for the five alternative scenarios applied to each DL model. In Figure 4.14b the *Unlimited Resources* scenario is plotted in red and should closely resemble the trend of the plot in Figure 4.14a that shows the



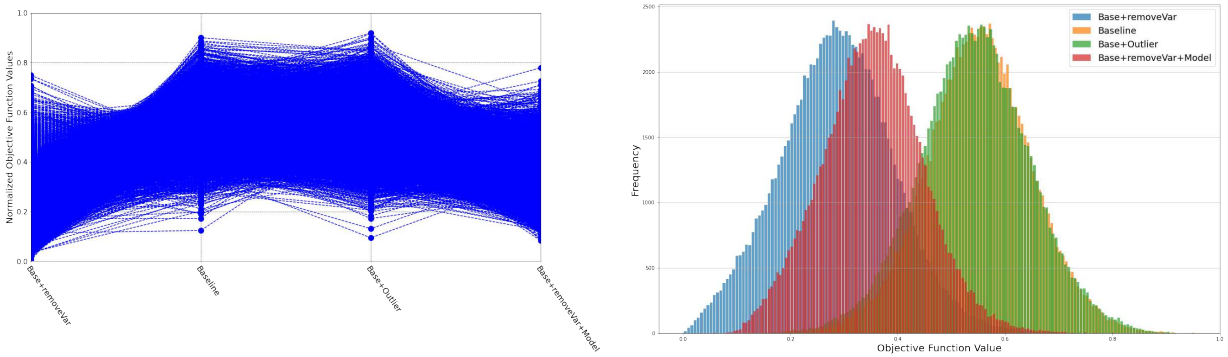
(a) MSE **(b)** AoA

Figure 4.14: MSE performance vs AoA relative objective function performance

relative performance based solely on MSE. This indicates that the *Base+RemoveVar* DL technique is still the best model for this scenario. Figure 4.14b also shows that the *Base+RemoveVar+Model* model is the best model for the second scenario where all models meet or surpass the system MSE tolerance. This indicates that when system costs are the driving factors and other areas of your system limit your accuracy, then ML sensor fusion models that require fewer resources are the best choice. The other scenarios indicate a similar trend that shifts the focus away from performance measurements alone for selecting DL techniques for sensor fusion. Figure 4.14b also shows a trend that each model has its own distribution of objective function values with the spread of different coefficients used in each scenario. This observation is shown in more detail with the Monte Carlo analysis which quantifies the relative performance distribution of these four models in any random scenario involving different weights of the objective function input variables. The results of the Monte Carlo analysis are presented in Section 4.5.2.3.

4.5.2.3 Monte Carlo Analysis

The Monte Carlo analysis begins where the AoA left off. The AoA only considered five different, but specific, scenarios, while this analysis is focused on the statistical quantities of 10,000 different random scenarios. This Monte Carlo analysis randomly varied the weight coefficients of each value in the objective function for these 10,000 different scenarios and plotted each scenario in Figure 4.15a. This figure shows a similar view as Figure 4.14b except it includes 10,000 sce-



(a) DL technique vs objective function

(b) Monte Carlo objective function distributions

Figure 4.15: Monte Carlo analysis of objective function coefficients for 10,000 runs

narios instead of just five. Figure 4.15b is a plot of the objective function distributions of these 10,000 scenarios for each of the four models. It is easy to see that the *Base+RemoveVar* model outperforms the other models on average when measured by the objective function. This happens to line up with comparing these four models on MSE performance alone as well. However, the *Base+RemoveVar+Model* model is a close second place on average which does not line up with MSE measurements alone. The other two models have almost identical distributions and may result in very little performance difference between the two if selected for a sensor fusion system.

4.6 Aircraft Case Study Discussion

The GD-MAGS process is one example of implementing an automated DevOps-like approach, or MLOps approach, for YAMM on a data system. The focus for this case study and the ML selection approach is to automate the ops feedback loop from the collection of operational data towards the continuous improvement of a legacy aircraft sensor fusion algorithm. This novel GD-MAGS process focused on a system of combining several different data preprocessing algorithms, Neural Network algorithms and hyperparameter tuning process together, rather than simply investigating a new method of Deep Learning algorithms alone. This selection approach could have continued to iterate using the MSE gradient after iteration five and further refined the hyperparameters, but we decided this minimum was sufficient for demonstration purposes. The value of this approach has been shown by its successful identification of the best DL technique and hyper-

parameter configuration. Part of this success can be shown by the removal of 6 DL techniques known to perform poorly. In one example, two under performing DL techniques were eliminated in the first iteration. The second worst technique, *Base+Outlier+Flightbatch*, only differed from the best technique, *Base+Outlier* by their respective batch sizes of 125,000 to 50,000 but had an 88% difference in average MSE as shown earlier in Figure 4.11. This matches observations made by Keskar et al. [137] that found there is a degradation in the quality of the model when using a larger batch size as measured by its ability to generalize.

In the second example, the selection approach pruned another four techniques when it correctly identified the poor performing *Model* procedure techniques. A key assumption used in this procedure was discovered late in this study to be wrong. We incorrectly assumed, initially, that the `RALT` measurement indicated the aircraft's distance above the earth, and by adding this value to the DTED terrain elevation it would provide better estimate of the altitude. Unfortunely, the range of the radar altimeter was limited to 5,500 feet. Therefore, the data provided by this procedure added meaningless and possibly misleading data to the LSTM algorithm which proved to affect the model's accuracy. The selection approach correctly pruned the DL techniques with this procedure and demonstrated value in removing an under-performing technique.

Two possible challenges of implementing this approach is balancing the correct number of bootstrap loops and number of flight files to use. Increasing bootstrap loops or files significantly slows down the selection process and increases compute resources required. Too few bootstrap loops or files introduces the risk of stochastic results or poorly generalized models. We verified our choices of bootstrap loops and number of files used in this demonstration by analyzing the average MSE for 20 bootstrap loops and doubling the number of test files for the selected model. This showed that the average MSE with three bootstrap loops vs 20 only differed by 1.04e-07. It also showed that the average MSE using 14 testing files, instead of the 7 done in this proposal, for 20 bootstrap loops only differed by 2.34e-08. These values are less than the difference between the best and second best model average MSE and indicates using 7 testing files for 3 bootstrap loops

was sufficient for this data set. However, we recommend users perform similar studies on their data.

Of note, the objective function used in this case study may not necessarily be the most ideal equation, however it fit our needs of demonstrating the value of the ML system analysis process. The results shown in the Monte Carlo distribution plots show that this equation and distribution of coefficients drastically favors the *RemoveVar* function out of the four models. This may indicate a bias towards these DL techniques with fewer variables, which end up affecting multiple other objective variables such as throughput, pre-processing, and volatile memory. However, the system analyst must decide what objective variables are important to their specific system and write the objective function accordingly.

4.6.1 ML Trade Study Analysis Discussion

Once again, the focus of this trade study analysis was not the implementations, per se, instead it was the demonstration of adopting a DevOps-like philosophy towards implementing YAMM. The manual creation of the SysML Model Card in this implementation goes against this philosophy and a future solution with the SysML 2.0 API specification may enable automation in the near future. However, the machine-readable SysML Model Card file enabled the ability to automate and evaluate the rest of the sensor fusion system analysis process. The current SysML version (1.6) has many advantages, however interfaces to SysML are currently tool dependent and not standardized. The updated SysML v2 language *beta* and SysML v2 API *beta* specifications could be foundational this effort by providing a standard machine-readable interface for system models [138]. Our novel application of machine readable ML Model Cards for system analysis is not intended to be the definitive solution, but we recognize that this standardization process is needed to enable future progress in this field and for YAMM. Our additions to Model Cards provide additional and future possibilities toward storing Model Cards in a data library for quick ingest and analysis. This scalability and automation will greatly improve the YAMM philosophy to automate the MDAO process with feedback from the operational system.

An ML Model Card library would make searching for models with certain characteristics for specific uses relatively easy. One could analyze this Model Card library to see where gaps in your modeling which could drive development of new and specific models towards second or third order and more accurate meta-algorithm models. If developed correctly and for very specific purposes, models could become analogous to software micro services that are optimized to perform very specific tasks. A Model Card catalog of these micro-service ML models and algorithms would enable a designer or developer to choose from and deploy models with accurate estimates of their performances and cost metrics. These ideas, while somewhat fanciful, need enabling technologies and we believe this extension to Model Cards, the automation for trade studies, and YAMM analysis and optimization automation will provide a step in that direction.

4.7 Aircraft Case Study Conclusions

We reiterate our objective of applying YAMM to this legacy aircraft data system here for convienience:

1 Demonstrate YAMM on a real-world legacy data system example to provide user insights into the methodology better than discussing its philosophies and general principles alone

2 Demonstrate the value of collecting operational data and automating empirically-based algorithm training and selection

3 Demonstrate YAMM's flexibility to train ML models for use in an OLTP data system

4 Demonstrate the value of YAMM's DevOps capabilities to integrate OLTP software development with system engineering system analyses on resource-limited data systems

5 Demonstrate an example of applying meta-algorithmics principles on a complex ML modeling problem such as sensor fusion

We met objective 1 throughout the demonstration of YAMM on this case study. This provided an OLTP system contextual example of YAMM implementation that would have been difficult to understand with our methodology description in Chapter 3 alone. Sections 4.7.1 and 4.7.2 discuss how we met our other objectives for this case study.

4.7.1 ML Selection Conclusions

The GD-MAGS selection process successfully demonstrated it is capable of selecting a single DL technique out of many while simultaneously tuning hyperparameters to generate an optimal sensor fusion model for a system of multi-modal sensors. This ML selection approach provides a reliable process that many engineers can apply to sensor data fusion problems with the knowledge it will converge on an optimal solution.

Our ML selection approach has also verified that its selected sensor fusion model generalizes well to other aircraft tail numbers with the same sensors. This verification provides a compelling justification that this ML selection optimization approach can be applied across many data sets and the selected optimal model can be applied to characteristically similar sensor fusion problems. The generalization of this approach can also apply to other ML selection problems outside of the sensor fusion discipline. The application of the GD-MAGS selection approach has successfully met objective 2 by demonstrating the value of YAMM's extension to collect ops data and automate the training and selection of empirically based algorithms that provide an accurate and well generalized ML model.

This approach enabled a DevOps software development mindset for the historically difficult problem of sensor fusion. This application also met objectives 3 and 5 above by training an accurate sensor fusion model for a legacy aircraft based on combining multiple preprocessing algorithms, a DL algorithm, and multiple methods of DL algorithm training. valuable insights into possible areas of advancement for sensor fusion and ML selection approaches.

4.7.2 ML Trade Study Analysis Conclusions

The results shown in Section 4.5.2 indicate accuracy measurements provide an adequate estimation of overall model performance for systems with infinitely scalable resources, however, arguably, cloud scalable resources are resource limited by a companies budget and should be considered similar to hardware resource limited systems. Resource-limited systems or simulations with timing constraints must consider both performance and resource usage cost measurements

when selecting ML models. This observation reinforces the need to consider downstream ops resources by performing ML trade study analyses when selecting ML models. ML algorithm development and performance will continue to accelerate and drive the need to provide a scalable and automated system analysis process. The application of this ML selection and analysis approach has shown that it benefits from standardized, machine-readable, and quantitative ML Model Cards in addition to machine-readable SysML system model specifications. This also demonstrates the value of implementing YAMM's DevOps extension to the Harmony aMBSE to ensure automation and collaboration between system engineers, software developers, and the operations teams provide downstream insight during the system design phase.

The novelty in this approach is the addition of system resource costs in machine readable RNN Model Cards for system analysis of sensor fusion models with the input of machine-readable SysML system specifications. These novel contributions are the enabling technology that supports rapid iterations and incremental improvements of sensor fusion algorithm development, system analysis and design, and general ML model implementation. This meets objective 4 of implementing YAMM on this legacy aircraft case study.

Chapter 5

Case Study 2: Hybrid Cloud Data System

In this chapter we demonstrate YAMM on the design, development, deployment, and optimization of a hybrid cloud data system prototype illustrated in 5.1. This case study reviews the steps we took to iterate through the YAMM process on the way to completing this prototype. The previous legacy aircraft use case utilized YAMM's stages 3 & 4, however this hybrid cloud case study demonstrates the entire methodology. These two case studies were not related, but we present them in a way to show the cost savings possible when applying YAMM to both an OLTP data system and its associated OLAP analytical data system.

Results from the aircraft case study reinforce the value of data collection and aggregation of both test and operational data from legacy systems into an OLAP system for analysis. After reflecting on the aircraft study, we realized access to high quality and easily available data in a modern data analytics environment (OLAP system) was a necessity for the modernization and upgrades for legacy DoD systems.

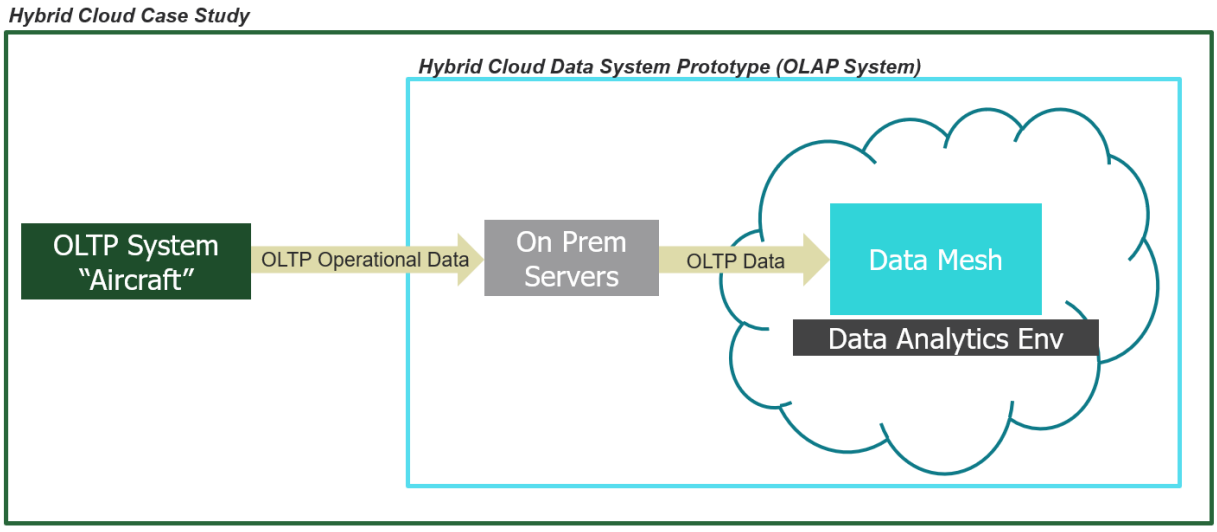


Figure 5.1: Aircraft Example for the Hybrid Cloud Case Study

Figure 5.1 illustrates the hybrid cloud case study in which the hybrid cloud data system prototype collects operational data from a notional aircraft OLTP system via on-prem servers. These on-prem servers automatically upload this data to the cloud where the ETL pipeline parses, translates, and loads data into the Data Mesh. This aggregated data is now qualified as analytical data and the Data Mesh enables data analysis from the analytics environment shown in the figure. The purpose of this hybrid cloud OLAP system is to enable users to train AI/ML algorithms, perform trend analysis, and execute any other arbitrary query or analytics on the analytical data towards supporting the OLTP aircraft example system. The initial Data Mesh in this case study only consists of a single Data Product. Subsequent development that increases the number of Data Products is outside the scope of the study.

The first step in building a large scale OLAP or HTAP system is to build a prototype for aggregation, normalization, and analysis of operational data. This OLAP system prototype must be capable of processing and storing aircraft data of multiple formats and also enable easy access to its data for analysts. In addition to optimizing costs and performance, the following list contains our objectives for this case study:

1 Identify the best recording format and communication protocols between three data formats
2 Identify the optimal ETL pipeline storage strategy between EBS and S3
3 Define the optimal data retention and cloud governance policies
4 Estimate on-prem CapEx and OpEx costs for the hybrid cloud system
5 Evaluate YAMM's ROI when reducing costs, balancing performance, and calculating the value of raw source after its been processed.
6 Demonstrate YAMM to provide insights into the application on an OLAP system.
7 Provide additional insights above those learned from the YAMM application to the the aircraft case study.
8 Demonstrate YAMM's value of incrementally and iteratively modeling, designing, developing, analyzing, and optimizing a data system from the beginning of initial design

9 Demonstrate the UMA and different meta-algorithms to capture system emergent behaviors and evaluate the model's accuracy.

10 Demonstrate the parallel nature of developing data system MVPs and YAMM's continuous analysis and optimization.

Therefore, we implemented YAMM during the design and development of this system prototype to build a financially responsible, high performance, and dependable OLAP data system. This prototype took 21 months to execute 4 YAMM iterations, 5 UMA iterations, 2 system optimization iterations, and 15 software versions deployed to the ops environment.

5.1 Case Study Overview

In this section we provide an overview of the hybrid cloud data system case study, the environments where we applied YAMM, the UM metamodel, the four hybrid cloud data system MVPs, a Work Breakdown Structure (WBS) for this case study, and the timeline of when all of this was accomplished. Additionally, we show the parallel effort of applying YAMM's continuous analysis and optimization during the development and deployment of the four MVPs. Later sections discuss each YAMM iteration in more detail and present the incremental results for each iteration.

Figure 5.2 is a SysML IBD of the designed environments to implement the *MDAO* and *Data Analytics & Algorithmics* stages for this hybrid cloud prototype case study. This figure shows the UM, the hybrid cloud prototype, the data analytics environment, and the datastore where the test, resource data, and system metadata are stored. It also shows the data flows between each of these environments and the SE, SW-Dev, and DI teams that work on the products. We used this design to setup the modeling environment, data analytics environment, and the datastore in the right locations to enable easy flow of information between these environment and the hybrid cloud prototype data system. The *Dev-V&V* and *Operational Data System* stages are executed within the *Hybrid Cloud Prototype : Data System* block shown in this figure.

Figure 5.3 shows this case study's UMA metamodel composed of the hybrid cloud UM libraries and the five versions of the UMs produced over the 4 YAMM iterations. The version numbers are

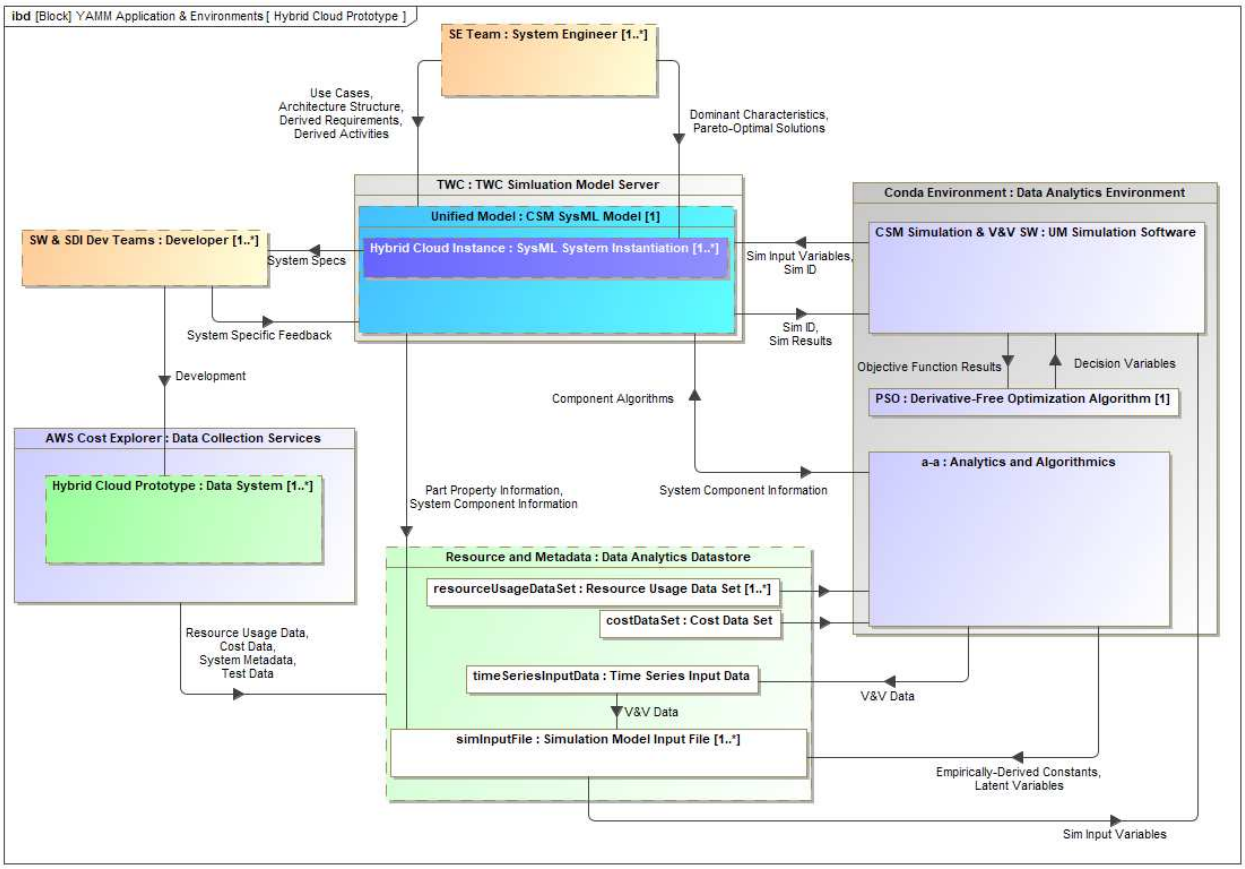


Figure 5.2: YAMM Implementation for the Hybrid Cloud Case Study

above each UM which progress in maturity from left to right. The hybrid cloud UM libraries are continually created or updated as the UMA incrementally adds capabilities to them. The dotted lines with arrows point to the model projects that are a *dependency of* the parent model project. These are called project usages. The darker lines from UM Ver.5 in this figure shows the final state of the model and library dependencies. Additionally, the colors of each model represent the lowest level of UM meta-model layer provided by the model as illustrated in 3.4, where red is level 1, cyan is level 2, and yellow is level 3. For instance, model version Ver.2 contains UM meta-model layers 1, 2, & 3 so it is colored red. Additionally, the model libraries shown in Figure 5.3 are built to be reused with future Agile Data System (ADS) models and will be continually updated.

Figure 5.4 illustrates the maturation of the hybrid cloud data system UM versions in the model repository. This figure also labels the level of abstraction for each model version as they mature and their fidelity increases. Model Ver.1 & Ver.2 were developed in parallel, where Ver.1 was purely

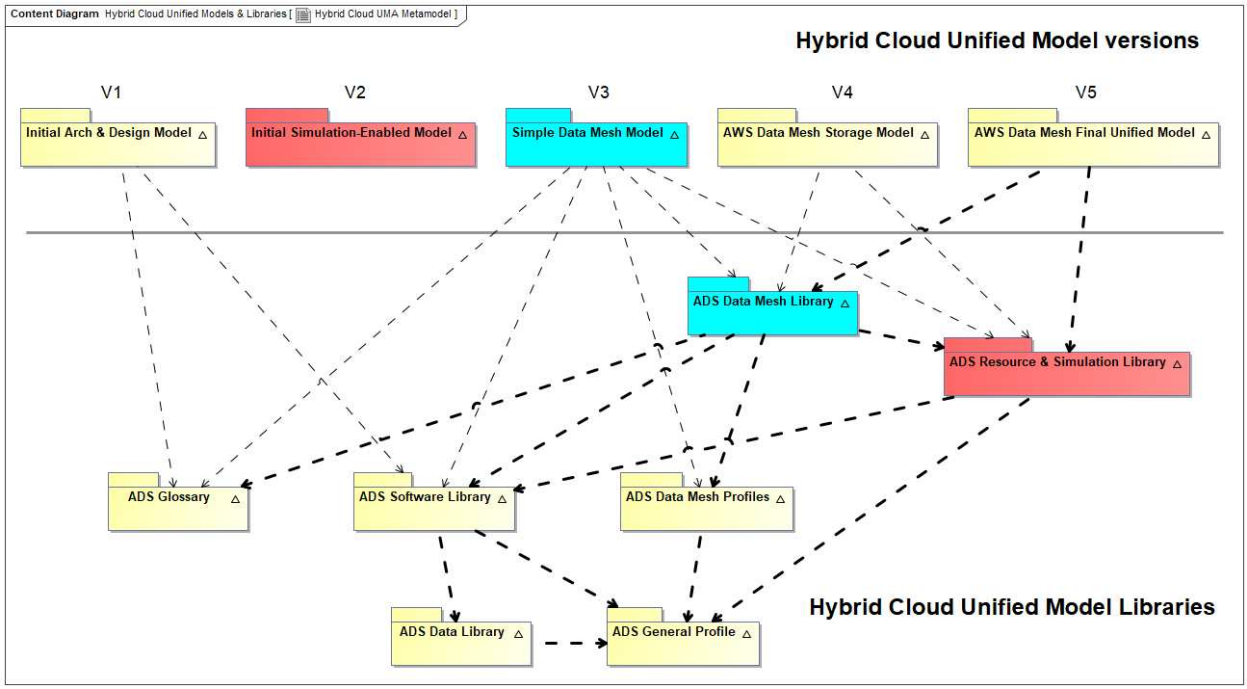


Figure 5.3: Hybrid Cloud Case Study UMA System and Library Metamodel

specifications and uses two specification level model libraries as shown in Figure 5.3. Whereas model Ver.2 was self contained with all three levels of specification and simulation, and didn't use any libraries during development. The lack of libraries for model Ver.2 is illustrated in Figure 5.3. Model Ver.3 was the first model built using all libraries. The additional libraries used by Ver.3 were built from separating model Ver.2 into modular pieces.

Figure 5.4 shows that model Ver.4 was the result of merging model Ver.1 & Ver.3 together, then adding capabilities. Model Ver.4 is modeled at the logical and physical levels of abstraction using specification and simulation elements inherited from the rest of the model libraries. Finally, model Ver.5 is the final model which was an update from Ver.4 in capabilities and in the amount of component behaviors modeled.

The following list contains a brief description of each of the four MVPs designed, developed, and deployed during this case study. The OLTP operational data has three different recording formats and communication protocols each corresponding to Interface Control Documents (ICDs) that describe these formats.

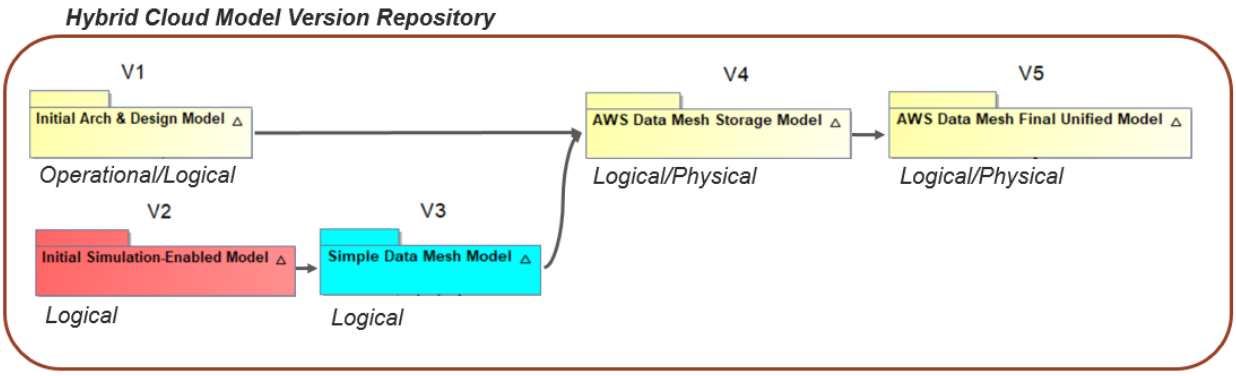


Figure 5.4: UM Version History and Levels of Abstraction

MVP 1: Hybrid cloud operational DI, Data Mesh environment, and the data analytic environment as illustrated in Figure 5.1. This includes the supporting DevSecOps DI and environments shown in Figure 3.17 to enable continuous development, testing, integration, security scans, and deployment of new data system MVPs.

MVP 2: A cloud native data lake-based Data Product within the Data Mesh that includes an automated ETL pipeline to ingest and parse data from format 1 and load it into the data lake catalog and datastore. This Data Product also includes a basic query service to enable users in the data analytics environment to find and access the data after it has been loaded into the Data Product.

MVP 3: Add a new ETL pipeline to the data lake-based Data Product to ingest and parse data from format 2 and load it into the same catalog and datastore as format 1.

MVP 4: Add a new ETL pipeline to the data lake-based Data Product to ingest and parse data from format 3 and load it into the same catalog and datastore as formats 1 & 2.

Figure 5.5 shows the case study schedule over 21 months and delineates YAMM iterations 1 through 4 along the top. The WBS is sorted into five categories which correspond to the four MVPs and the continuous YAMM analysis and optimization work. Additionally, each WBS item is colored by the YAMM life cycle stage from which it was executed. The colored YAMM stage legend is at the bottom of this figure.

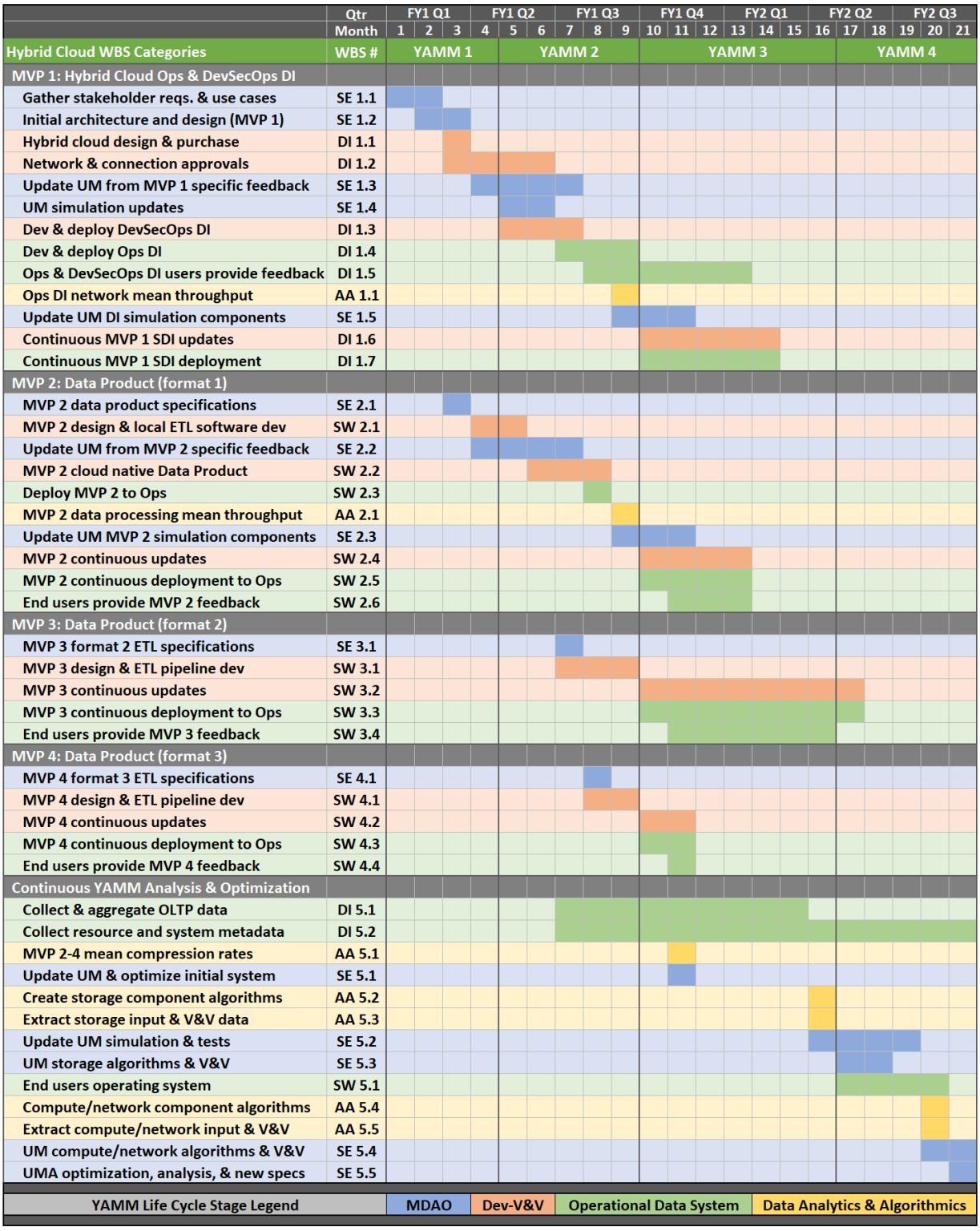


Figure 5.5: Hybrid Cloud YAMM WBS Schedule

In Figure 5.5, the WBS numbers are labeled by team and by WBS category. For example, SE 1.2 is the second WBS item assigned to the SE team in the first WBS category. The SW represents the SW-Dev team, the DI represents the DI team, and the AA represents the Analytics & Algorithmics (AA) team. We broke this schedule into MVPs to show the parallel development of YAMM as well as the continuous analysis and optimization that happens throughout a data system's life cycle.

Several WBS tasks shown in Figure 5.5 overlap YAMM iteration boundaries. This illustrates that DevSecOps and YAMM incremental and iterative updates do not always start and stop at the same time while parallel MVPs are being worked. For instance, WBS number DI 1.2 is a long lead item for network and connection approvals and therefore the rest of the YAMM stages do not need to wait for it to finish before they start the next MVPs if they are not blocked. The continuous and iterative DevSecOps cycles in YAMM iterations 2, 3, & 4 that deploy software and SDI from the *Dev-V&V* stage to the *Operational Data System* stage are also shown in this figure in WBS pairs DI 1.6/1.7, SW 2.4/2.5, SW 3.2/3.3, and SW 4.2/4.3. During DevSecOps cycles for MVPs 2, 3, & 4 there were 15 different software versions continuously deployed to the operational data system.

Tables 5.1, 5.2, 5.3, 5.4, and 5.5 list each WBS item, its description, and the associated step in the YAMM Process Flow 1 for the five different WBS categories, respectively.

Sections 5.2, 5.3, 5.4, and 5.5 summarize the tasks performed and their results during YAMM iterations 1-4, respectively. Section 5.6 discusses the continuous YAMM analysis and optimization which includes a matured UM that has undergone V&V. Section 5.7 presents the results of the optimization and analysis of this matured UM. These sections cover the time frames shown in Figure 5.5 and the WBS tasks happening in parallel. The conclusions from applying YAMM to this case study are in Section 5.8.

Table 5.1: Hybrid Cloud DI (MVP 1) WBS and YAMM Process Steps

WBS #	WBS Description	YAMM Process Flow 1 Step #
SE 1.1	Multidisciplinary team meets with stakeholders, gathers requirements & use cases, and iterates with feedback in parallel with SE 1.2.	1.1
SE 1.2	SE team chooses Data Mesh architecture, builds initial structural design w/requirements & use cases for MVP 1, & provides feedback to SE 1.1. UM ver 1.0 is released and accessible to everyone.	1.2 & 1.3
DI 1.1	DI team designs cloud and on-prem hardware, then purchases on prem hardware, cloud space, and an internet package.	2.1 & 2.1.1
DI 1.2	DI team works on network ATO, software approvals, and network connection approvals using the UM and other artifacts.	2.1.1
SE 1.3	SE team updates the UM specifications from the software and DI designs.	1.3
SE 1.4	SE team develops the initial UM simulation functions, utilities, & tests.	1.4
DI 1.3	DI team continuously develops and deploys the DevSecOps DI & SDI.	2.2 & 2.3
DI 1.4	DI team continuously develops and deploys the operational DI & SDI.	2.2 & 3.1
DI 1.5	SW-Dev team provides feedback about the DevSecOps DI & SDI while end users provide feedback about the operational DI.	3.2 & 2.4
AA 1.1	The AA team calculates mean network throughput from the on-prem servers to the cloud using the S3 bucket resource data collected by AWS.	4.1
SE 1.5	The SE team updates the UM simulation of the DI based on feedback from the SW-Dev & DI teams and analytics from the AA team.	1.4
DI 1.6	The DI team has their DSOP setup and continuously develop, test, scan, and deliver the DevSecOps SDI and the operational system SDI.	2.2& 2.7
DI 1.7	The DI team continuously deploys the fully tested and scanned DevSecOps SDI and the operational system SDI to their respective environments.	2.3

Table 5.2: Data Product w/format 1 (MVP 2) WBS and YAMM Process Steps

WBS #	WBS Description	YAMM Process Flow 1 Step #
SE 2.1	The SE team defines the cloud native Data Product specifications and delivers it with the format 1 ICD specifications.	1.3
SW 2.1	The SW-Dev team designs the Data Product, then starts to develops the ETL software to parse & translate format 1 data locally while waiting for the DevSecOps environments to be ready.	2.1 & 2.4
SE 2.2	The SE team updates the UM specifications based on feedback from the SW-Dev team Data Product design.	1.3
SW 2.2	After the cloud-base DevSecOps DI is deployed from MVP 1, then the SW-Dev team develops the rest of the Data Product, builds the DSOP, and delivers the software to the operational environment.	2.4 & 2.7
SW 2.3	The SW-Dev team deploys the initial MVP 2 Data Product to the operational data system environment.	3.1
AA 2.1	The AA calculates the mean throughput of the ETL pipeline for format 1 of the Data Product and sends this to the SE team.	4.1
SE 2.3	The SE team updates the UM Data Product simulation components based on feedback from the SW-Dev & AA teams.	1.4
SW 2.4	The SW-Dev team continuously develops, tests, and scans the Data Product software for MVP 2 and delivers it to the operational environment.	2.4
SW 2.5	The SW-Dev team continuously deploys the Data Product software to the operational data system environment.	3.1
SW 2.6	The end users in the operational environment provide feedback to the SW-Dev team.	3.2

Table 5.3: Data Product w/format 2 (MVP 3) WBS and YAMM Process Steps

WBS #	WBS Description	YAMM Process Flow 1 Step #
SE 3.1	The SE team delivers the data format 2 ICD to the SW-Dev team.	1.3
SW 3.1	The SW-Dev team designs and develops the data format 2 ETL pipeline & DSOP to integrate with the MVP 2 Data Product.	2.1 & 2.4
SW 3.2	The SW-Dev team continuously develops, tests, and scans the data format 2 ETL pipeline and delivers the software to the operational data system.	2.4 & 2.7
SW 3.3	The SW-Dev team continuously deploys the data format 2 software to the operational data system environment.	3.1
SW 3.4	The end users in the operational environment provide feedback to the SW-Dev team.	3.2

Table 5.4: Data Product w/format 3 (MVP 4) WBS and YAMM Process Steps

WBS #	WBS Description	YAMM Process Flow 1 Step #
SE 4.1	The SE team delivers the data format 3 ICD to the SW-Dev team.	1.3
SW 4.1	The SW-Dev team designs and develops the data format 3 ETL pipeline & DSOP to integrate with the MVP 3 Data Product.	2.1 & 2.4
SW 4.2	The SW-Dev team continuously develops, tests, and scans the data format 3 ETL pipeline and delivers the software to the operational data system.	2.4 & 2.7
SW 4.3	The SW-Dev team continuously deploys the data format 3 software to the operational data system environment.	3.1
SW 4.4	The end users in the operational environment provide feedback to the SW-Dev team.	3.2

Table 5.5: Continuous YAMM Analysis & Optimization WBS and YAMM Process Steps

WBS #	WBS Description	YAMM Process Flow 1 Step #
	Continuous YAMM Analysis & Optimization	
DI 5.1	The end users offload Aircraft (AC) (OLTP system) data to the on prem network which uploads data to the cloud.	3.2
DI 5.2	AWS services continually measure and collect resource and cost data and make this available with system metadata in AWS *Cost Explorer*.	3.3
AA 5.1	The AA team calculates a mean compression rate for each of the three data formats for MVPs 2-3 using the S3 bucket resource data.	4.1
SE 5.1	The SE team adds ETL compression rate constants to the UM simulation capability and input files, then optimizes an simple Data Mesh system using a PSO algorithm to create a Pareto-front for cost and performance. cost and performance objective functions.	1.4 & 1.4.2
AA 5.2	The AA team identifies AWS data system storage structure and matching UM storage components, models storage behavior, identifies latent variables, classifies data sources and throughput, and parameterizes storage costs.	4.1
AA 5.3	The AA team creates V&V model storage & cost inputs for initial conditions, time series inputs, time series results, and latent variable constants.	4.1
SE 5.2	The SE team updates UM specification & simulation behaviors, increases model fidelity, and adds storage & cost latent variables to UM.	1.3 & 1.4
SE 5.3	The SE team creates a UM simulation V&V instantiation, sets up UM input files using V&V data, creates & runs V&V tests, then fixes the UM defects until it passes.	1.4.1
SW 5.1	The end users operate the full system with all MVPs functioning and provide feedback to the SW-Dev team.	3.2
AA 5.4	The AA team identifies all other components of AWS costs and models their behaviors using latent variables, factoring, and other analytics.	4.1
AA 5.5	The AA team creates V&V model inputs of: compute, network, recurring, and purchase costs for: initial conditions, time series, and latent variable constants. Also creates time series results for measuring accuracy	4.1
SE 5.4	The SE team updates the UM physical specification & simulation behaviors and structure to model the rest of the AWS costs.	1.4
SE 5.5	The SE team creates & runs V&V tests, fixes defects until they pass, then performs a system optimization and analysis.	1.4.1 & 1.4.2

5.2 YAMM Iteration 1

In this section we step through the first YAMM iteration, the tasks accomplished on the WBS, the work done in each YAMM stage, and the data passed onto the teams in the next stage. This being the first iteration, we gathered the initial stakeholder requirements and use cases, from which we chose a data system architecture, designed MVPs 1 & 2, started development on MVPs 1 & 2, and began the procurement, connections, and approvals for the DI.

5.2.1 *MDAO*

Within the first stage of the YAMM process, we met with the data system stakeholders, formed a geographically separated team of experts, then discussed and provided feedback about the data system requirements and use cases during WBS SE 1.1. Once we understood the stakeholder needs we chose the Data Mesh architecture for its flexibility and modularity since it is a decentralized data system architecture. Figures 5.6 and 5.7 show SysML diagrams of the stakeholder use cases, requirements, and test cases. These figures and the other figures in this section are from the Ver.1 model as shown in Figures 5.3 and 5.4. For this case study we focus our YAMM demonstration and initial software development MVPs on the ability to *Automatically Ingest "Aircraft" Data into a Cloud-Based Data System*, which is what we will call our primary use case to keep the demonstration more concise.

After selecting the system architecture we used the UMA, as discussed in Section 3.1.1, for WBS SE 1.2. We considered the scalable and modular system simulations needed for this system and chose SysML as the baseline visual DSL to use within our UMA because it is currently the most common and actively developed language for MBSE within the USAF. Unfortunately, SysML is not currently defined to sufficiently create a time-dependent and executable model, therefore we developed a combination of SysML, fUML, Apache Groovy, and Action Language Helper (ALH) to accomplish the scalable and modular simulation capability we need. We chose the Dassault Systemes (DS) line of tools based on commonality of tools with other organizations we interface with and ones that also support the languages we picked. These choices also considered

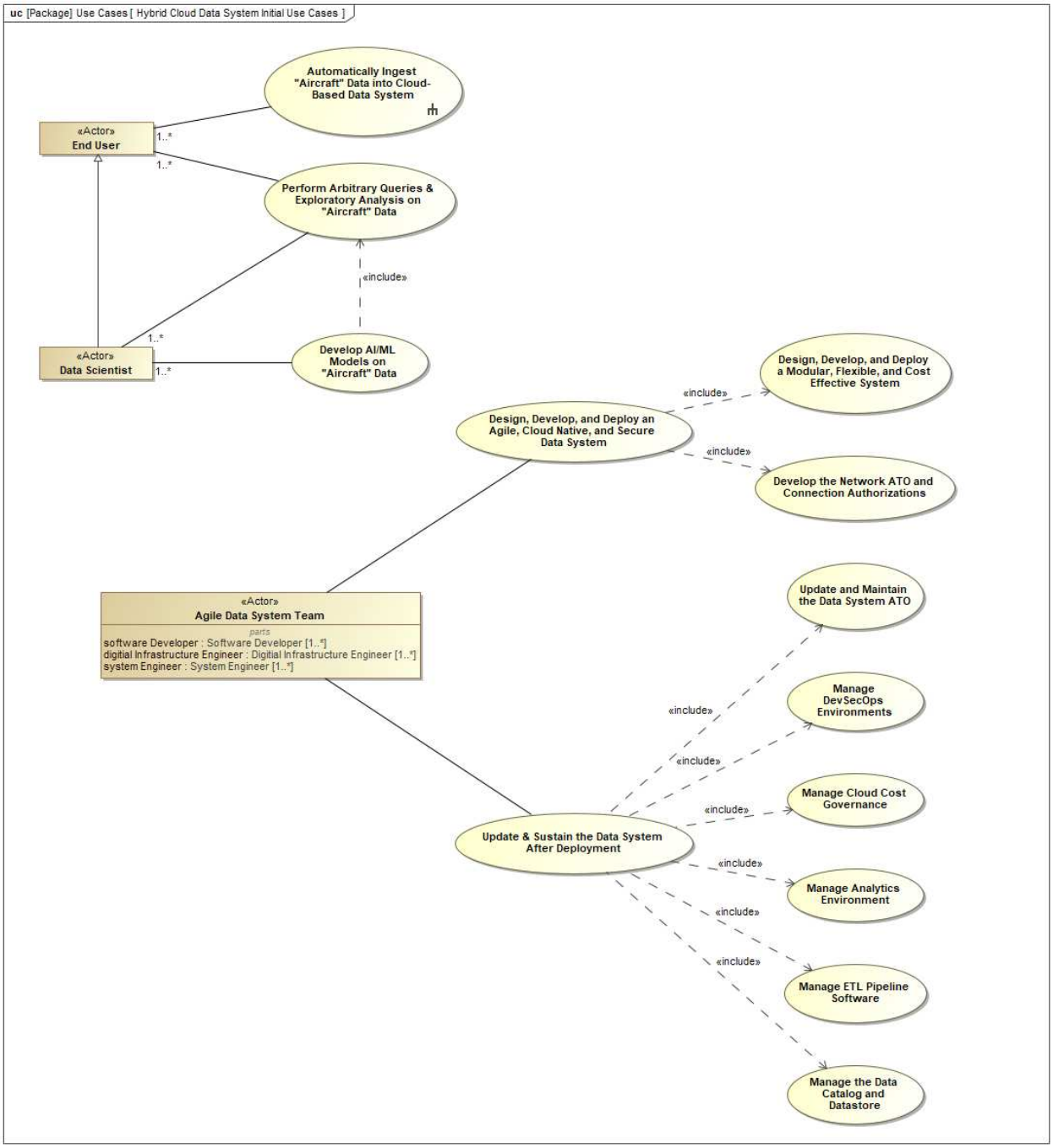


Figure 5.6: Stakeholder Use Cases (Model Ver.1)

the popularity and commonality of the DS suite of tools and languages to enable us to share these models with other organizations. However, the choice in tools and languages best suited for our YAMM and UMA approach may change with release of the SysML 2.0 specification and other tools that support it and simulation capabilities.

#	Name	Text	Verified By
1	8 DoD Data Quality Dimensions	Data owners shall categorize data sets at different levels of quality over time because of age, prioritization, initial condition, and other factors. Therefore, data quality dimensions are relative, and owners will assess the dimensions across the data's lifecycle.	
2	8.1 Data Accuracy	Data shall correctly reflect proven, true values or the specified action, person, or entity. Accuracy includes data structure, content, and variability.	Measure Data Format Compliance Frequency Measure Frequency Data Values Match Ground Truth Measure Data Error and Compare Against Error Tolerance
3	8.2 Data Completeness	The data shall present at a specified time contain the expected information or statistics, as measured at the data set, row, or column level.	Measure and Identify Null Values in the Dataset Fields Measure Dataset Breadth of Information for Contextual Purpose Measure Amount of Known Data That Would Make the Dataset More Complete
4	8.3 Data Set Conformity	Data sets shall follow agreed upon internal policies, standards, procedures, and architectural requirements.	Measure Data Format To The Applicable Standard Identify and Locate Dataset Published Architecture Measure and Identify Policies, Standards, and Producedures to Prevent Erroneous Input
5	8.4 Data Consistency	Data shall contain values that are uniformly represented within and across data sets.	Measure Frequency of Other Datasets That Reference Values in This Dataset Measure Discrepancies in Values Referenced From Other Dataset to Values in this Dataset
6	8.5 Data Uniqueness	Data shall be tagged to ensure there is a one-to-one alignment between each observed event and the record that describes such an event.	Identify If There Are Other Authoritative Data Sources That Serve The Same Function As This Data Source Measure The Size and Quantity of Duplicate Records in This Dataset.
7	8.6 Data Timeliness	Each type and source of data shall ensure the time between an event occurring and the data's availability for use is met.	Measure the Dataset Update Latency Against Data Consumer's Need Identify and Measure the Dataset Update Frequency Required by Data Consumers
8	8.7 Data Integrity	A data set's pedigree, provenance, and lineage shall be known and aligned with relevant business rules.	Measure Accuracy of Data Cleaning Process Measure Frequency of Data Quality Checks Identify and Measure Data Integrity During Collection, Storage, Processing, and Transmission
9	9 Cloud Native Software	Software running in the data system shall be cloud native.	
10	10 Authority to Operate	The data system shall obtain an Authority to Operate before it can begin to aggregate data from the OLTP system.	
11	11 Secure Software	Software shall be approved and added to the Approved Program List before it is allowed in the data system environments.	

Figure 5.7: Stakeholder Requirements and Derived Test Cases (Model Ver.1)

Figures 5.8 and 5.9 show the operational level of abstraction of the system architecture IBD and the behavior of this case study's demonstration use case, respectively. Figure 5.8 shows the sub-systems within and the external system to the hybrid cloud data system. The external systems are denoted with a dashed outline. This figure also shows the data flows between each sub-system and external system. Figure 5.9 shows the behaviors and the sub-system or external system they are allocated to. Additionally, this figure also shows the items flows between each activity. These item flows between behaviors are derived from the data item flows between the structural elements shown in Figure 5.8.

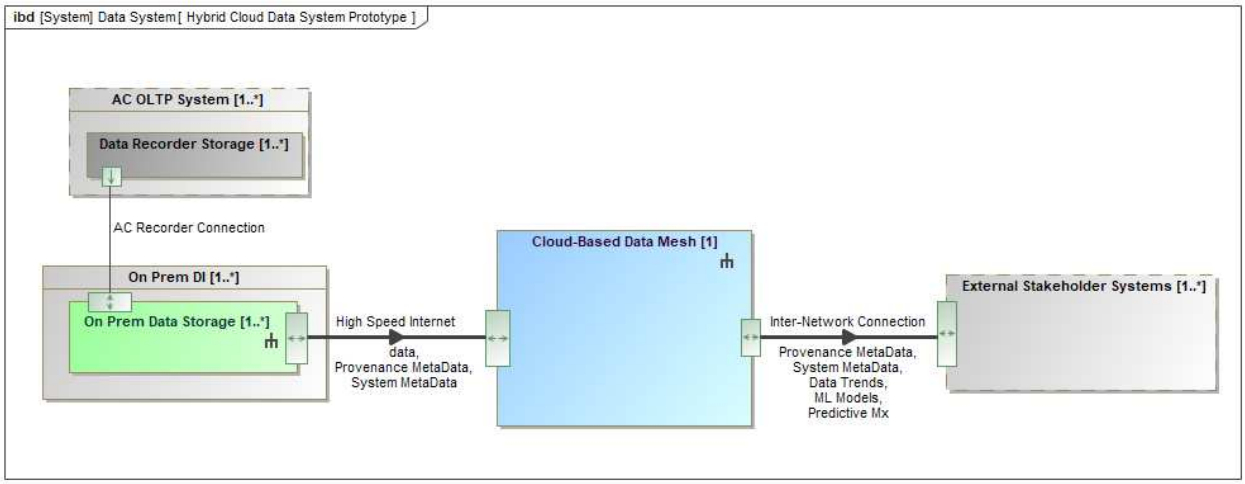


Figure 5.8: Agile Data System Operational Viewpoint (Model Ver.1)

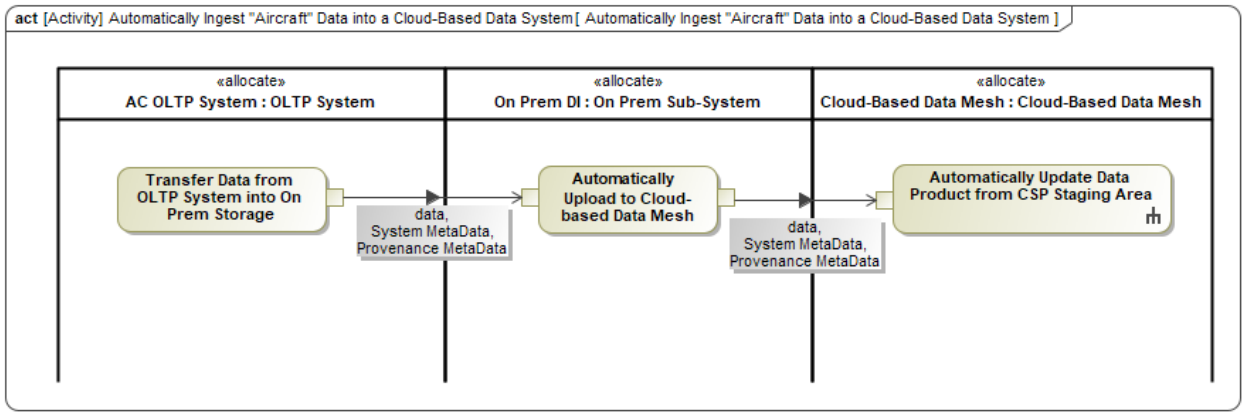


Figure 5.9: Automatic Ingestion of Data into Hybrid Cloud Data System (Model Ver.1)

Figures 5.10 and 5.11 show the structural and behavioral specifications for the cloud-based Data Mesh, respectively. Figure 5.10 shows the DevSecOps SDI, the development and test environments, and the DSOP for each software component of the Data Product being developed for this case study. This figure also shows the flow of development software from the *Dev Env* to the *Test Env* and deployment of this software to the Data Product. Additionally, this figure shows the flow of OLTP data (*data*) being uploaded to the cloud and loaded into the Data Product. Figure 5.11 shows the behavior specification of the software that automatically performs data ETL for the Data Product.

Figures 5.12 and 5.13 show the abstract data and metadata definitions and specializations for the OLTP and OLAP systems for this case study. The *Data ICD* definition shown in Figure 5.12

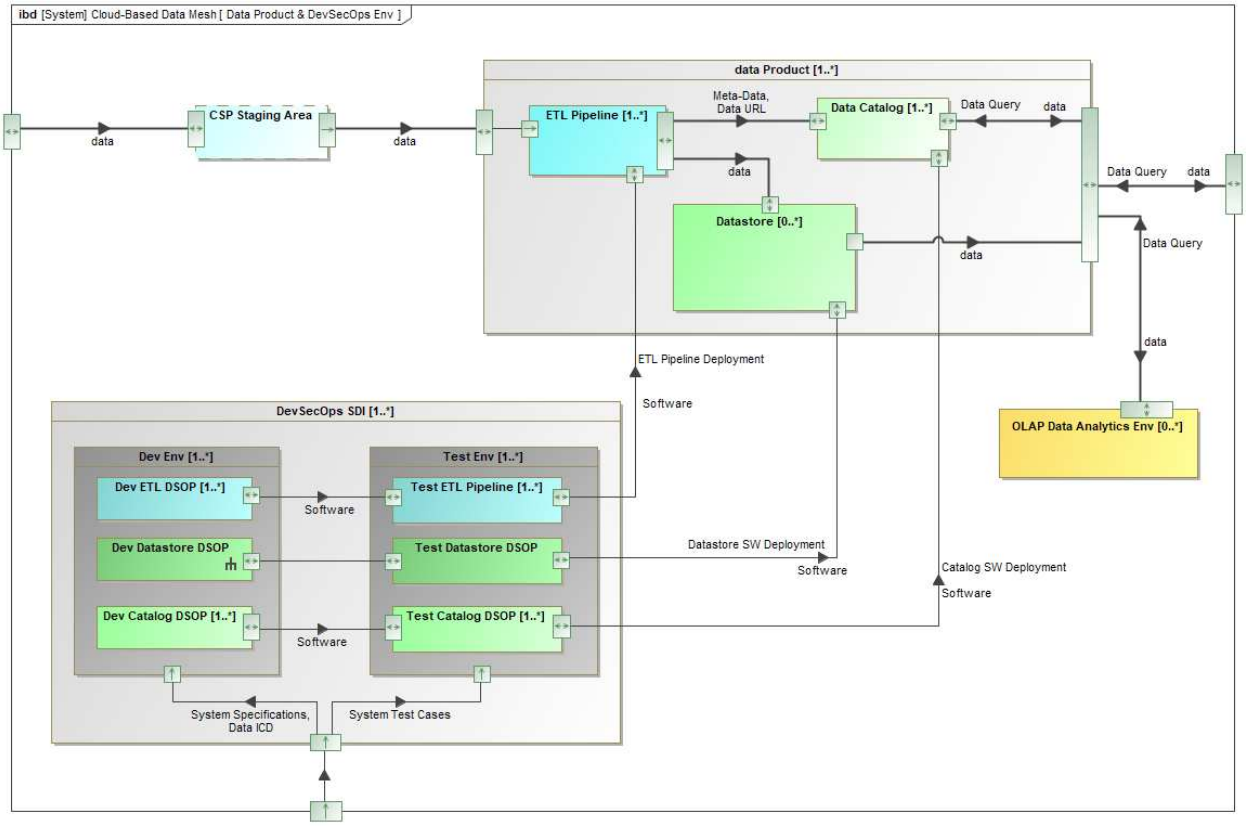


Figure 5.10: Data Product and DevSecOps Environment Specifications (Model Ver.1)

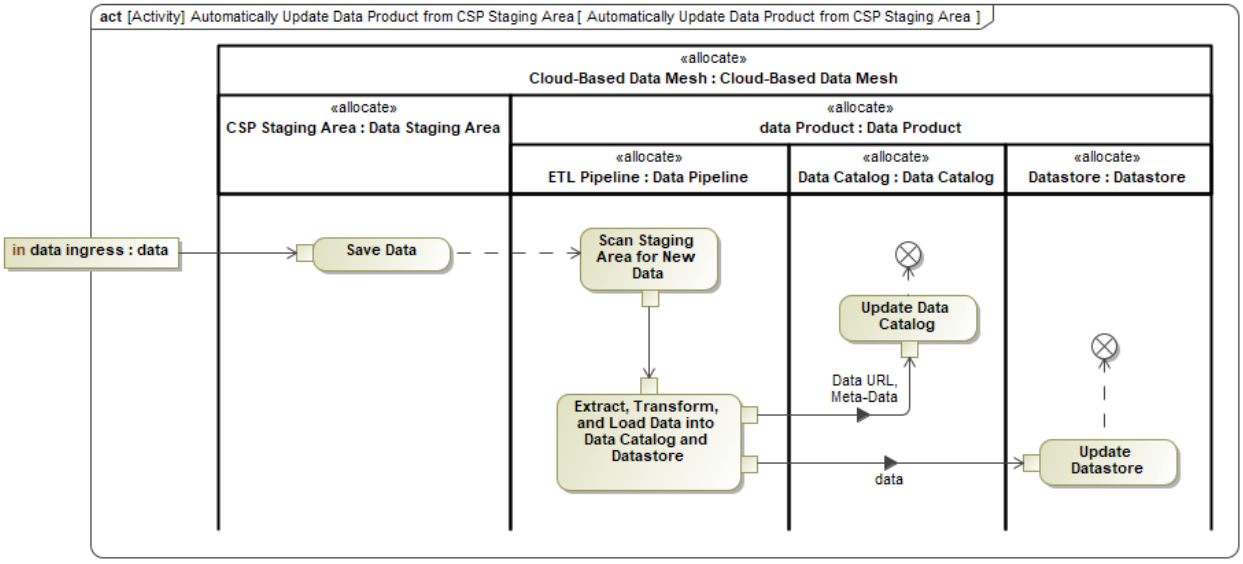


Figure 5.11: Data Product ETL Behavior Specifications (Model Ver.1)

is the definition of the ICDs that are required for MVPs 2 through 3. To ETL data into the Data Product the SW-Dev team requires a recording specification and a translation specification. For

example, an aircraft may record its avionics data in the IRIG 106 Chapter 10 (Ch10) recording standard [38]. This avionics data usually contains video of the Pilot Vehicle Interface (PVI), 1553 [139] serial bus traffic, or flight test instrumentation. Within 1553, data there are Command Words, Status Words, and Data Words that are 20 bits each. The Data Words need to be translated to a specific data type. These data types include the transactional data needed to fly the aircraft such as aircraft position, altitude, speed, etc. Therefore, an ETL data pipeline requires the the Ch10 recording specification and the 1553 translation specification to accurately populate the Data Product.

The use cases, requirements, system architecture, the structural and behavioral specifications, data format 1 ICD, and the test cases were delivered to the SW-Dev and DI teams for downstream engineering and sub-system design.

5.2.2 *Dev-V&V*

After the system specifications and test cases are delivered to the SW-Dev dna DI teams, they designed MVPs 1 and 2 and purchased the on-prem hardware, the internet service, and the cloud space to begin development of the SDI. The DI team designed the deployment of the SDI for the DevSecOps and operational data system environments via IaC for reusability. The SW-Dev team designed the cloud native software to be deployed to Kubernetes (K8s) clusters for scalability. These two teams had to collaborate closely since using K8s affects the definition of the SDI which must be defined in the IaC. These downstream engineering designs were fed back to the SE through sketches and several meetings.

The these teams collected the architecture, system specifications, and downstream design artifacts to include in the paperwork process of getting the ATO and network connection approvals. The SysML diagrams provided value to this process through the use of consistent and detailed data system ATO artifacts.

The SW-Dev team was able to begin development on the MVP 1 extraction and transformation software used in the ETL data pipeline. This was done on a local network while they were wait-

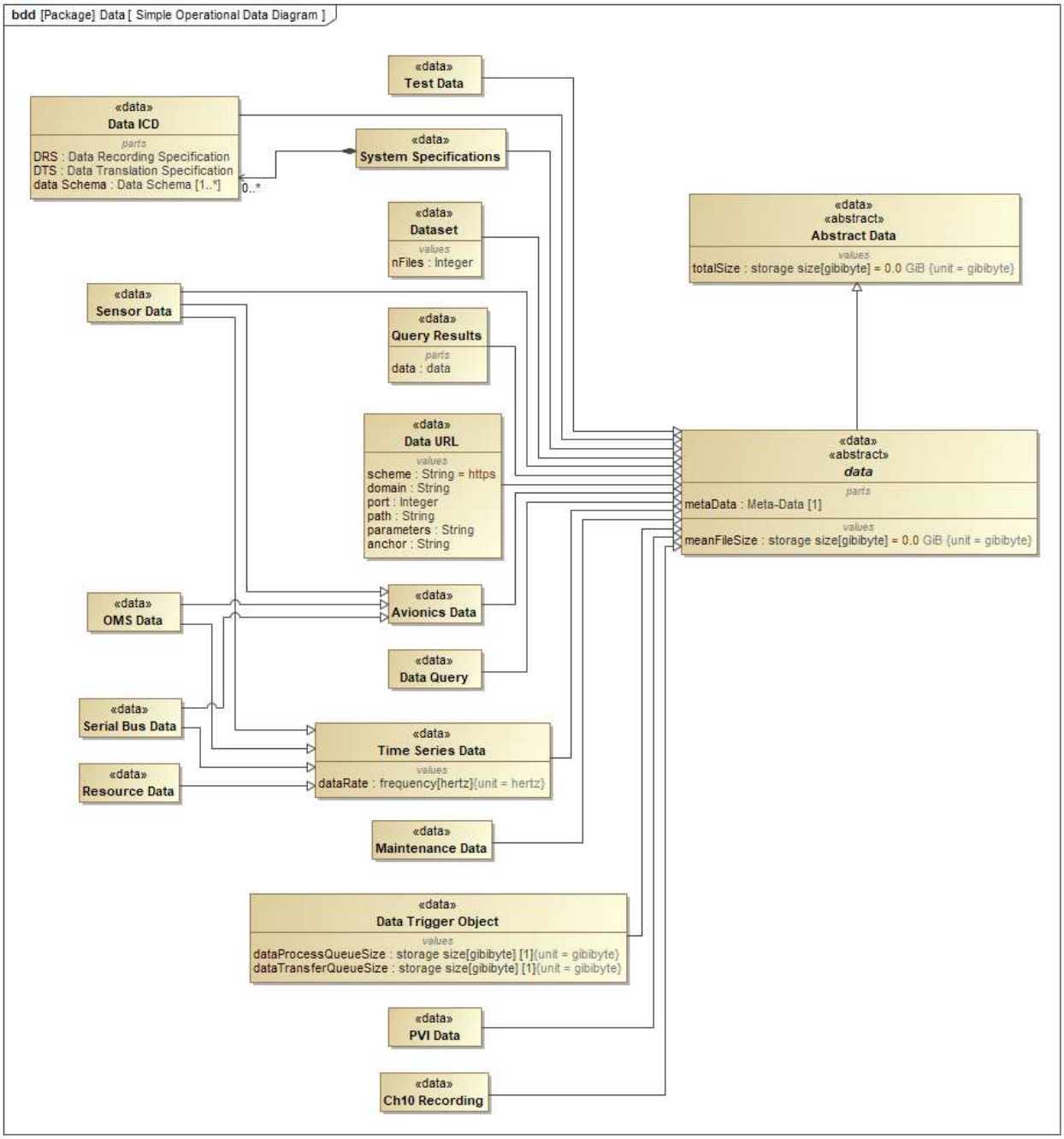


Figure 5.12: Abstract Data and Data Specialization Specifications (Model Ver.1)

ing for the DevSecOps environments to be deployed. This required the recording and translation specification ICDs from the data format 1 vendor and recorded sample data to verify functionality.

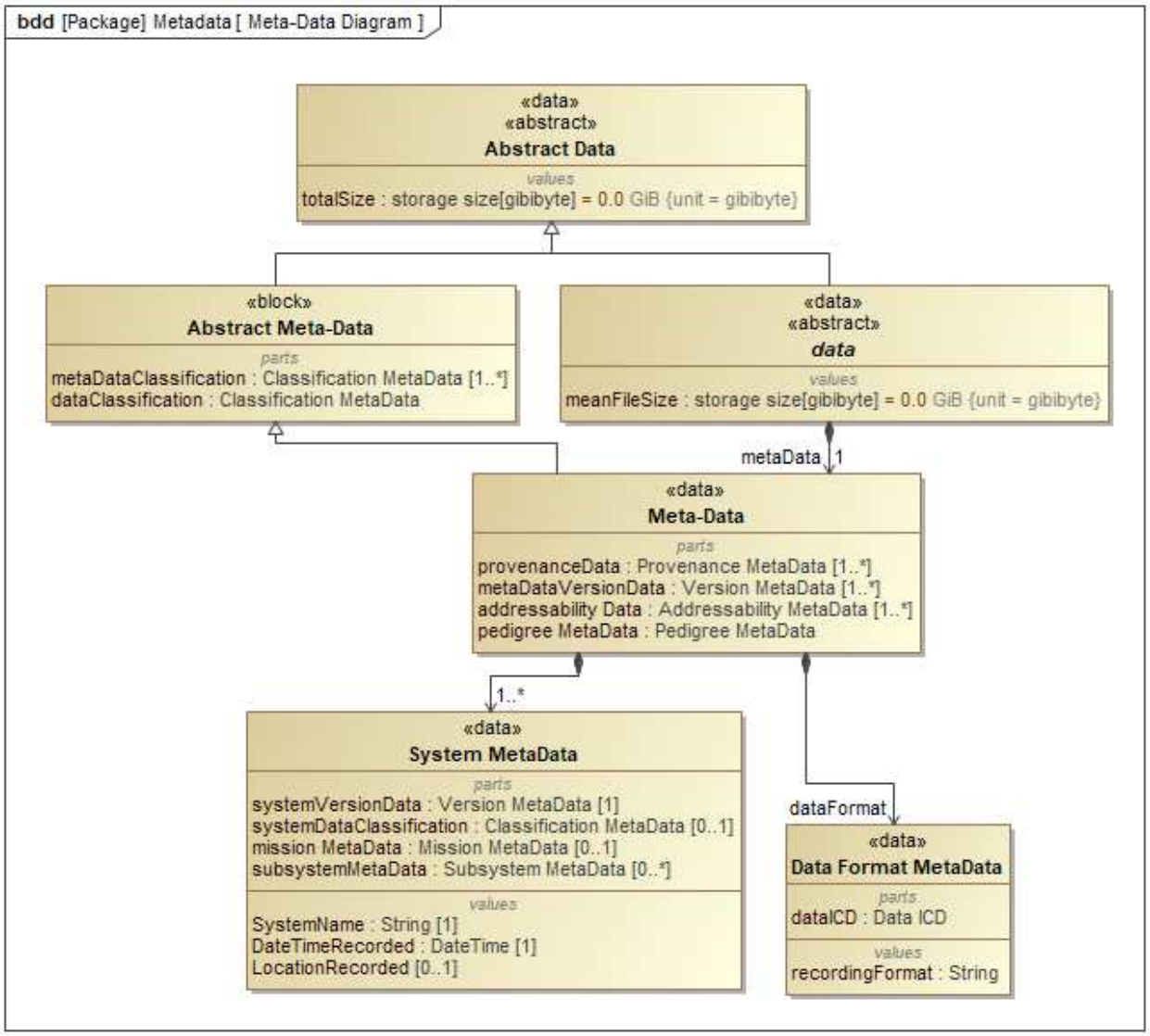


Figure 5.13: Abstract Metadata and Metadata Specialization Specifications (Model Ver.1)

5.2.3 *MDAO* Update

The SE team incorporated the downstream engineering design and system specific feedback into into the Ver.1 model as logical and physical levels of abstraction. This model is shown in reference to the Ver.4 model in Figures 5.3 and 5.4. The Ver.4 model inherits many of the logical and physical modeling elements from the Ver.1 model.

Figure 5.14 shows the IBD which contains much of the feedback from the downstream engineering teams. This figure shows a local server, Snowball, and a Daqscribe as three different

on-prem DI solutions needed for this prototype. The *recorder* from the aircraft can connect to the local server or an AWS Snowball, while the Daqscribe is both a data recorder and a server. The Snowball is an offline data transfer device that is shipped to an AWS location, connected to their servers, then data is uploaded into your AWS account. These Snowballs were used as the network connections were being approved and installed. The Daqscribe was used as a data collection method for other flight test experiments. Each of these three on-prem solutions have been defined to include automated data transfer pipelines to upload data to the cloud environment.

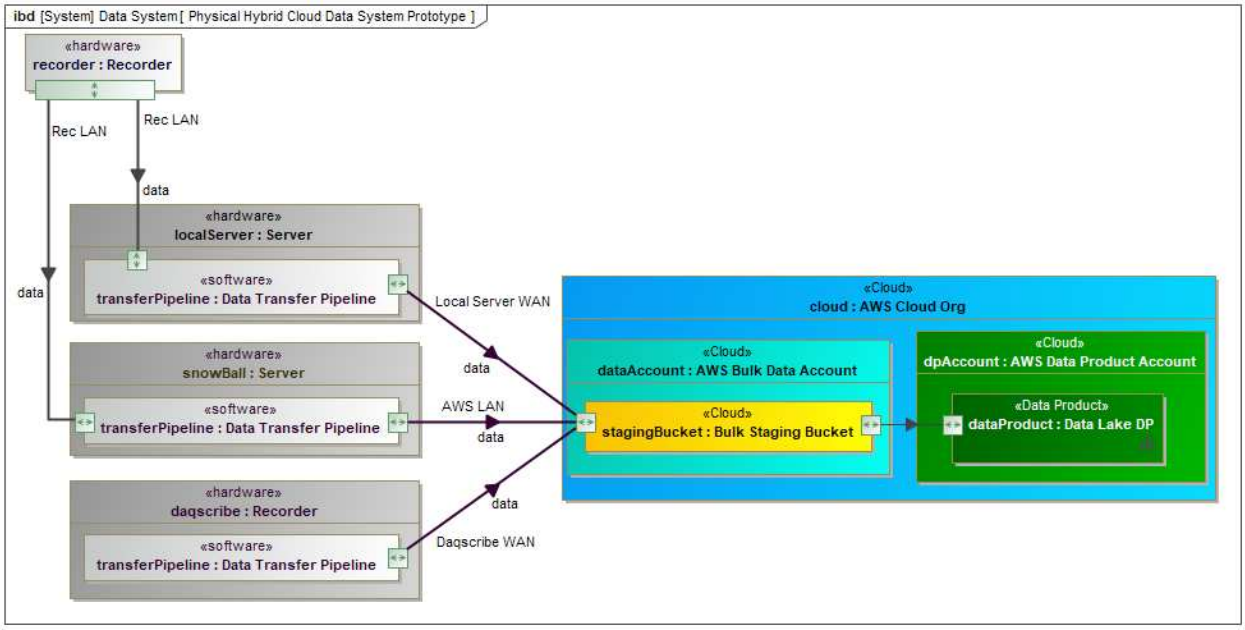


Figure 5.14: Hybrid Cloud Data System Physical Architecture (Model Ver.1)

Figure 5.14 also shows the SDI storage configurations which added the cloud staging bucket and the Data Product to their own AWS accounts. These account are contained within the AWS cloud the DI team purchased. This represents a typical AWS cloud storage structure and the different accounts enable resources and costs to be tracked separately.

The datalake-based Data Product shown in Figure 5.15 was also upgraded based on feedback from the downstream engineering teams and from additional research done by the SE team. This

figure shows the addition of a data publish service which is the front end of the Data Product that end users or other Data Products interact with.

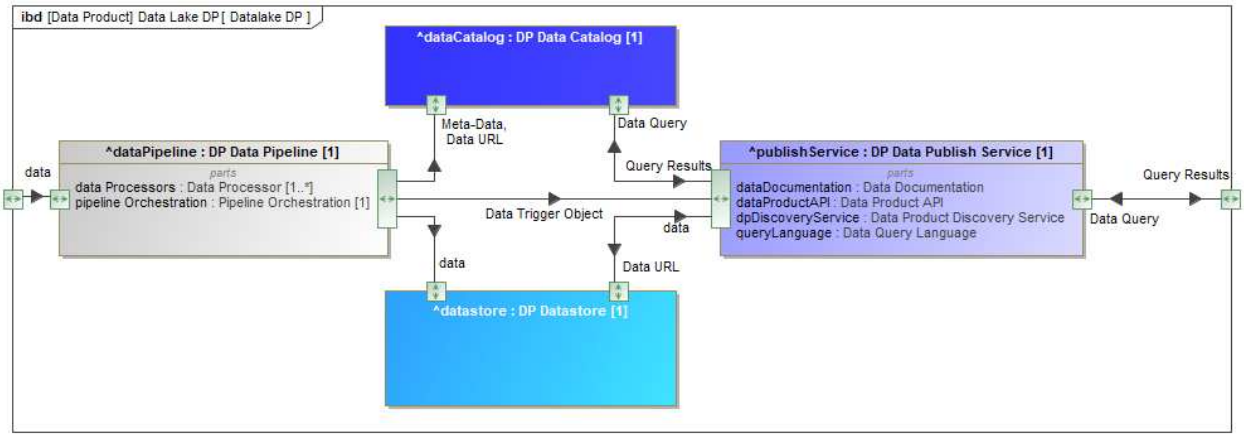


Figure 5.15: Datalake-Based Data Product Logical Definition (Model Ver.1)

The primary use case of this case study, shown in Figure 5.6, is decomposed into sub-system use cases and component behaviors. These sub-system use cases and component behaviors are shown in Figure 5.16. These were decomposed in the way described in Section 3.1.1.3 and represented by the illustration in Figure 3.5.

The activity diagram at the bottom of Figure 5.16 shows the component behaviors in the swim lanes of an activity partition. This is the method we use with the UMA to allocate behaviors to structural elements to ensure the execution context is correct during the simulation. This allocation is also discussed in Section 3.1.1.3 and illustrated in Figure 3.5.

The feedback from the downstream engineering teams continues to provide the SE team with useful insight into the system specific knowledge. The work to incorporate these insights and physical system attributes into the UM continues into the next YAMM iteration discussed in the next section.

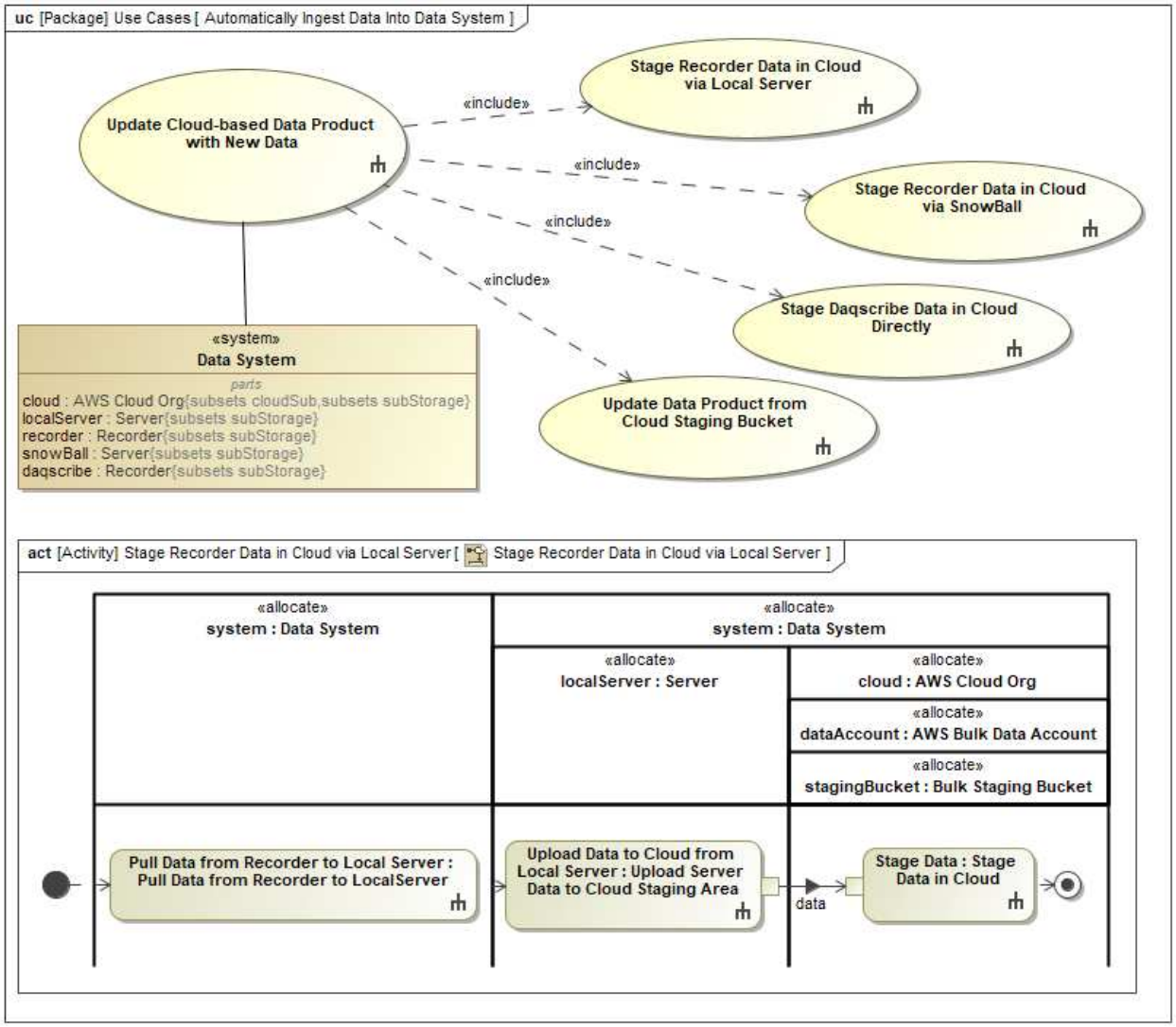


Figure 5.16: Primary Case Study Use Case and Activities Decomposed (Model Ver.1)

5.3 YAMM Iteration 2

In this section we discuss the updates and results of the UM versions, the progress of MVPs 1 & 2, the start of MVPs 3 & 4, the aggregation of operational data from the aircraft OLTP system, the collection and analysis of initial resource and system metadata. In this incremental YAMM update to the data system the SE team updates the specification model based on the feedback from the SW-Dev and DI which aligns with the YAMM Process Flow 1 step 1.3. In parallel, the SE team begins to work on the simulation enabled data system model for step 1.4.

We discuss the creation of an initial simulation-enabled UMs in Section 5.3.1 through the implementation of the UMA. This section summarizes the results of applying the UMA and is organized into subsections by the three UMA metamodel layers, simulation scenarios, simulation results, a discussion about the results, and our conclusions for this initial model. We summarize the development of MVPs 1 through 4 in Section 5.3.2. In Section 5.3.3 we provide an overview of the deployments of MVPs 1 & 2 to the operational data system, the collection of OLTP data, and the collection of initial resource data and system metadata from the operational data system. Lastly, we review the work done by the AA team to analyze and quantify initial data analytics in Section 5.3.4.

5.3.1 *MDAO*

In this section we present the initial simulation-enabled UM, the UMA metamodel layers that it contains, and the simulation results showing emergent system behavior using different system scenarios [25]. This is the Ver.2 model shown in Figure 5.4 and was developed in parallel with the Ver.1 model. The purpose of starting with a simple model is to develop small simulation capability MVPs to provide value quickly. Updates to the model can add new capabilities and each update is verified as the model becomes more complex.

5.3.1.1 MDAO: UMA Metamodel Visual Specification Layer

Figures 5.17, 5.18, 5.19, and 5.20 show the abstract subsystem block definition, the system definition, internal block diagram, and activity diagram of the initial simulation-enabled UM of a simple ETL pipeline system, respectively. This model is only focused on simulation of the primary use case of automatically ingesting OLTP data into a hybrid cloud data system. These figures are a logical abstraction of a simplified version of the hybrid cloud system. These figures show the UMA metamodel layer 3 visual specification elements referenced in Figure 3.4.

Figure 5.17 shows the abstract subsystem block that contains the common specification and simulation value properties, referenced datasets, and ports defined by general data interface to enable simulation execution. All other blocks inherit these definitions and add to them as seen in

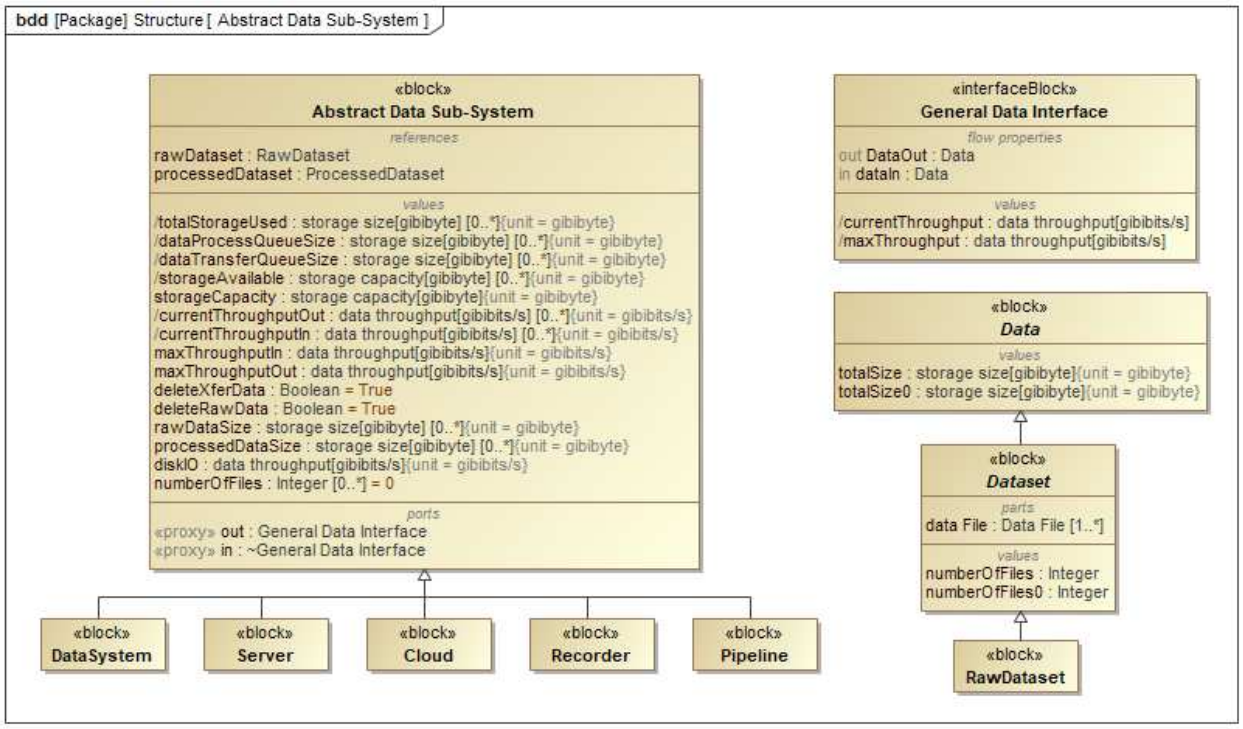


Figure 5.17: Abstract Sub-System Definition (Model Ver.2)

Figure 5.18. During simulation these value properties are inputs to the simulation algorithms and capture the outputs from these algorithms. For instance, the *deleteXferData* value property in the *Data Sub-System* block is a system specification to ensure data is deleted after its transferred out of the sub-system. This boolean value is read and changes the simulation behavior accordingly. This figure also shows the simulation value properties which are written to from simulation functions and utilities. For example, the *totalStorageUsed* value property keeps track of the amount of data stored in that specific block as times series historic data over the entire simulation. All blocks in this figure inherit from the *Data Sub-System* block. These simulation value properties and relation to the simulation functions and utilities are illustrated in Figure 3.6 and discussed in the UMA in Section 3.1.1.3.

5.3.1.2 MDAO: UMA Metamodel Visual Simulation Layer

Figures 5.21 and 5.22 show the simulation analysis block definition and its behavior diagram, respectively. The purpose of the simulation analysis block and its behavior is discussed in Sec-

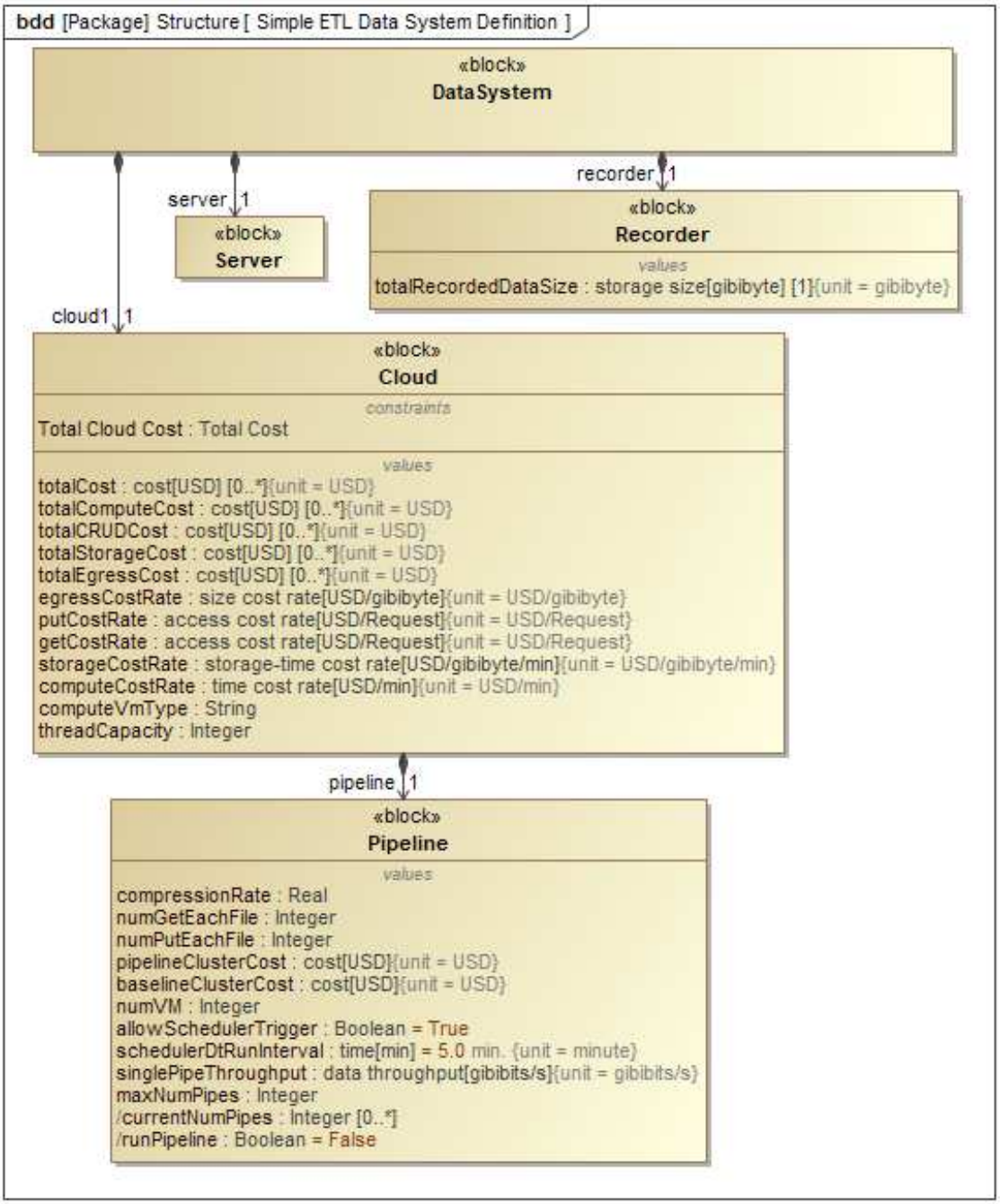


Figure 5.18: Simulation-Enabled Simple ETL Pipeline System Definitions (Model Ver.2)

tion 3.1.1.3 and illustrated in Figures 3.5 and 3.6. These define the use case simulation parameters and data system initialization.

Figure 5.23 shows an example of a simulation behavior as illustrated in Figure 3.6. This is the simulation definition of the `Transfer Data From Recorder` component behavior from the

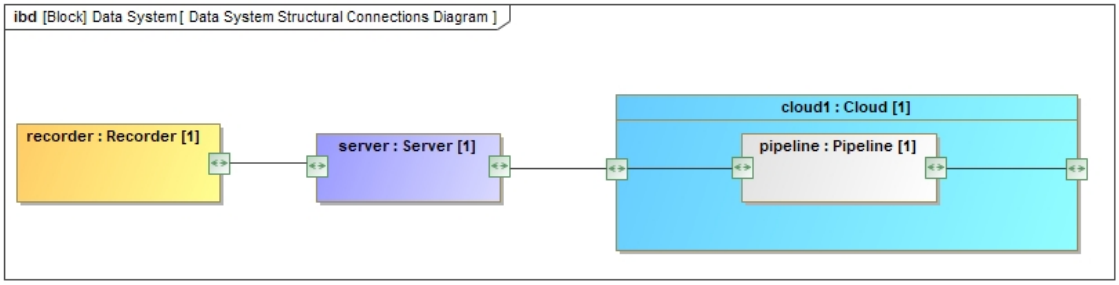


Figure 5.19: Simulation-Enabled Simple ETL Pipeline System IBD (Model Ver.2)

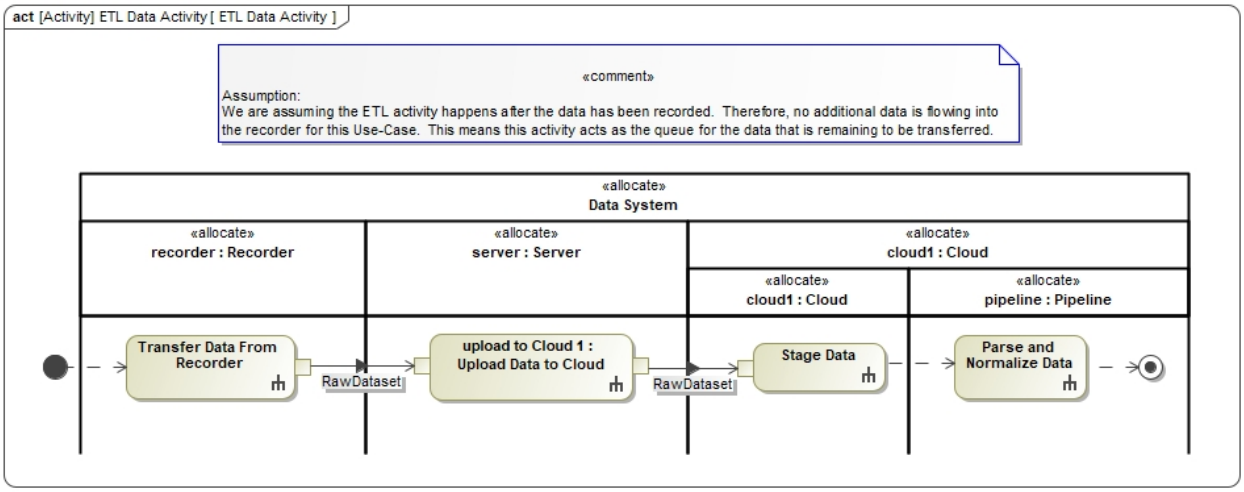


Figure 5.20: Simulation-Enabled Simple ETL Pipeline System SysML Activity Diagram (Model Ver.2)

ETL pipeline system behavior definition shown in Figure 5.20. The activity diagram in Figure 5.23 shows an example of fUML providing simulation execution context via the *«readSelf»* behavior. This behavior gets the execution context by which structural element it is allocated to while it is executing. In this example it is allocated to the *recorder* shown in Figure 5.20.

Access to a block's proxy port interface value properties is one limitation of using the execution context within SysML and the DS tool set. Unfortunately, we were not able to get the max throughput out of a block's port interface to calculate the amount of data that can flow through that during a single time step. Therefore we created a parametric diagram shown in Figure 5.24 to bind the General Interface's *maxThroughput* value to the Sub-System's max throughput in and out values.

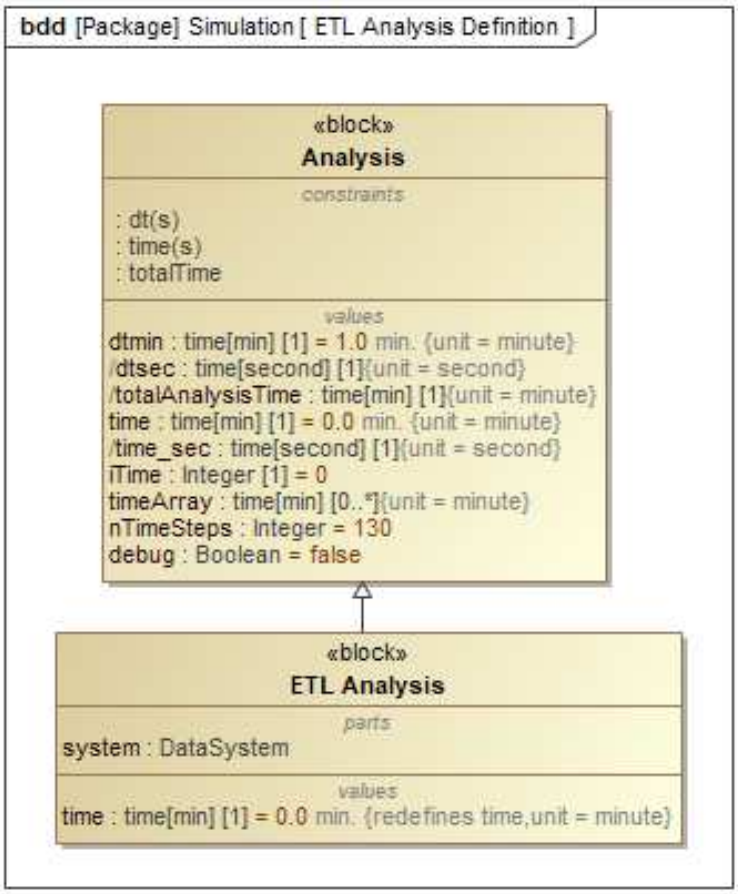


Figure 5.21: BDD of the ETL Analysis Block (Model Ver.2)

We also had to constrain the max throughput values of each subsystem within the data system so they were equal as shown in Figure 5.25. This ensured that a data throughput out of one subsystem couldn't be higher than the max throughput into the next system. This enables the opaque behavior *transferDataOut* in Figure 5.23 to calculate the amount of data to transfer in a single time step.

5.3.1.3 MDAO: UMA Metamodel Simulation Functions & Utilities Layer

An example of visual simulation element calling simulation functions and utilities is shown in Figure 5.26. This figure shows the definition of the *transferDataOut* opaque behavior that is shown in Figure 5.22.

The opaque behavior groovy code in Figure 5.26 shows that it gets the *ANALYSISOBJ* and *globalSimClass* objects by using an ALH command. The *ANALYSISOBJ* object is the *ETL Analysis* block shown in Figure 5.21 which provides the simulation variables to keep all simulation

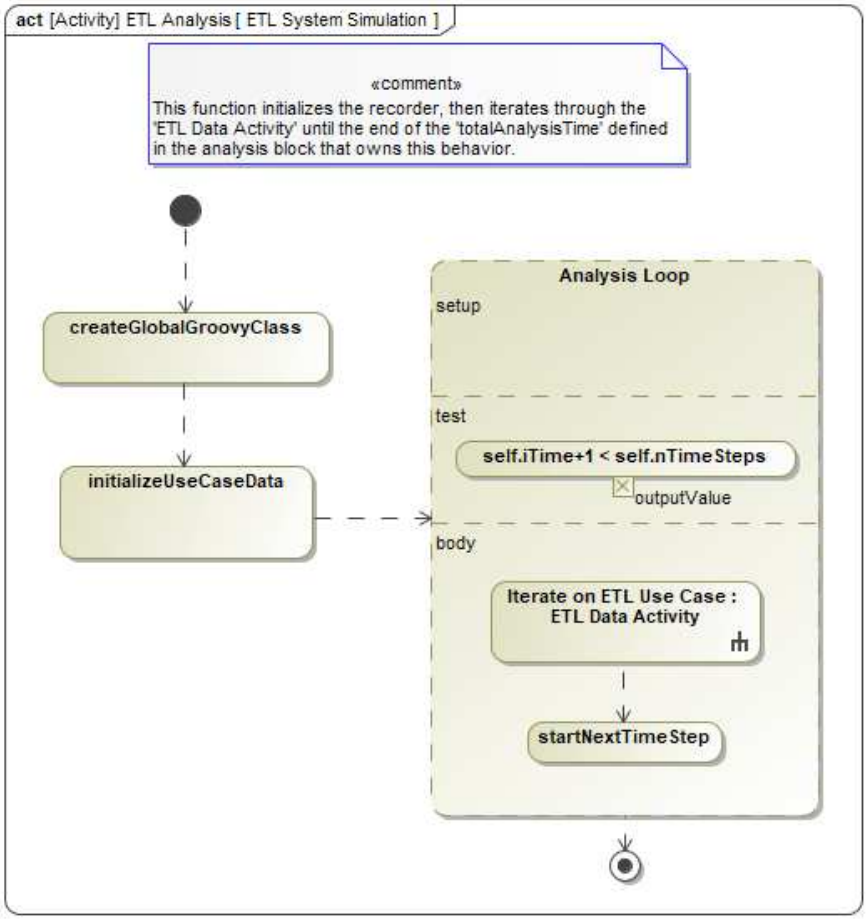


Figure 5.22: Activity diagram of the ETL Analysis Block (Model Ver.2)

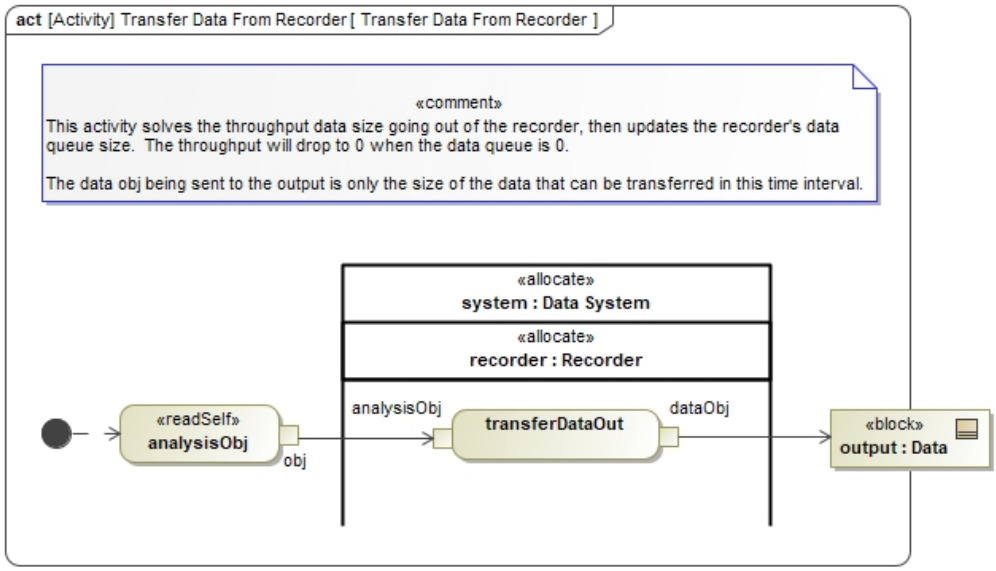


Figure 5.23: Simulation Behavior of the `Transfer Data From Recorder` Activity (Model Ver.2)

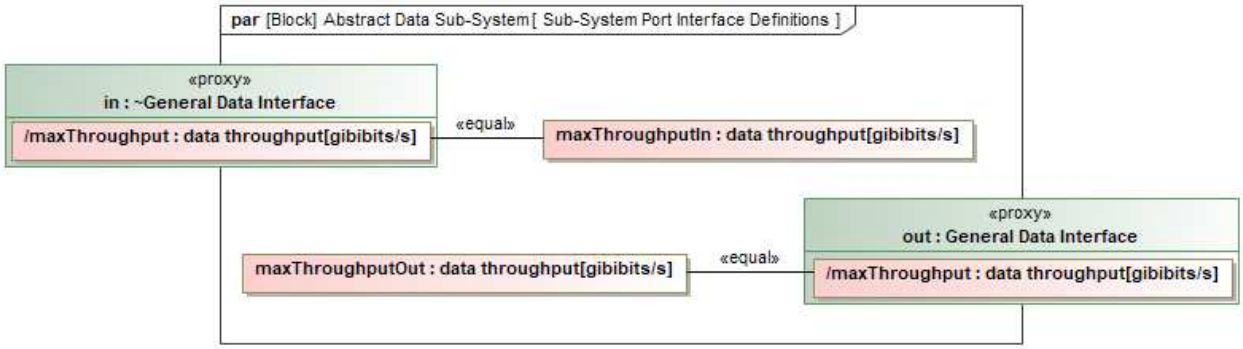


Figure 5.24: Sub-System Block Interface Definition (Model Ver.2)

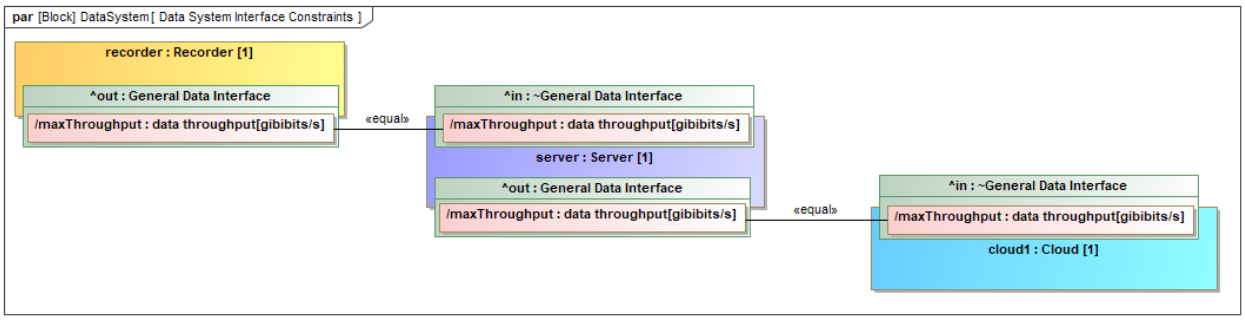


Figure 5.25: Data System Interface Definitions (Model Ver.2)

blocks in sync. The *globalSimClass* object contains all of the simulation functions and utilities. Both of these objects are created and set to global ALH variables in the *createGlobalGroovyClass* opaque behavior shown in Figure 5.22. The *checkListInit* and *solveLANThroughput* methods in this Class are used by the *transferDataOut* opaque behavior to calculate the size of data leaving the recorder during a time step. This data size is dependent on the *maxThroughputOut* recorder variable which is constrained as shown in Figures 5.24 and 5.25.

The opaque behavior in Figure 5.26 sets several *recorder* value properties through use of the execution context. The *dataTransferQueueSize* and *currentThroughputOut* are simulation arrays that are both used in this simple algorithm to keep track of transfer queues from the recorder and the data throughput values for each step in this simulation. This simulation process is illustrated in Figure 3.6 and discussed in Section 3.1.1.3.

The use of opaque behaviors, parametric diagrams, and Groovy methods to enable simulation for this simple ETL data system appears overly complex, however these modular capabilities were

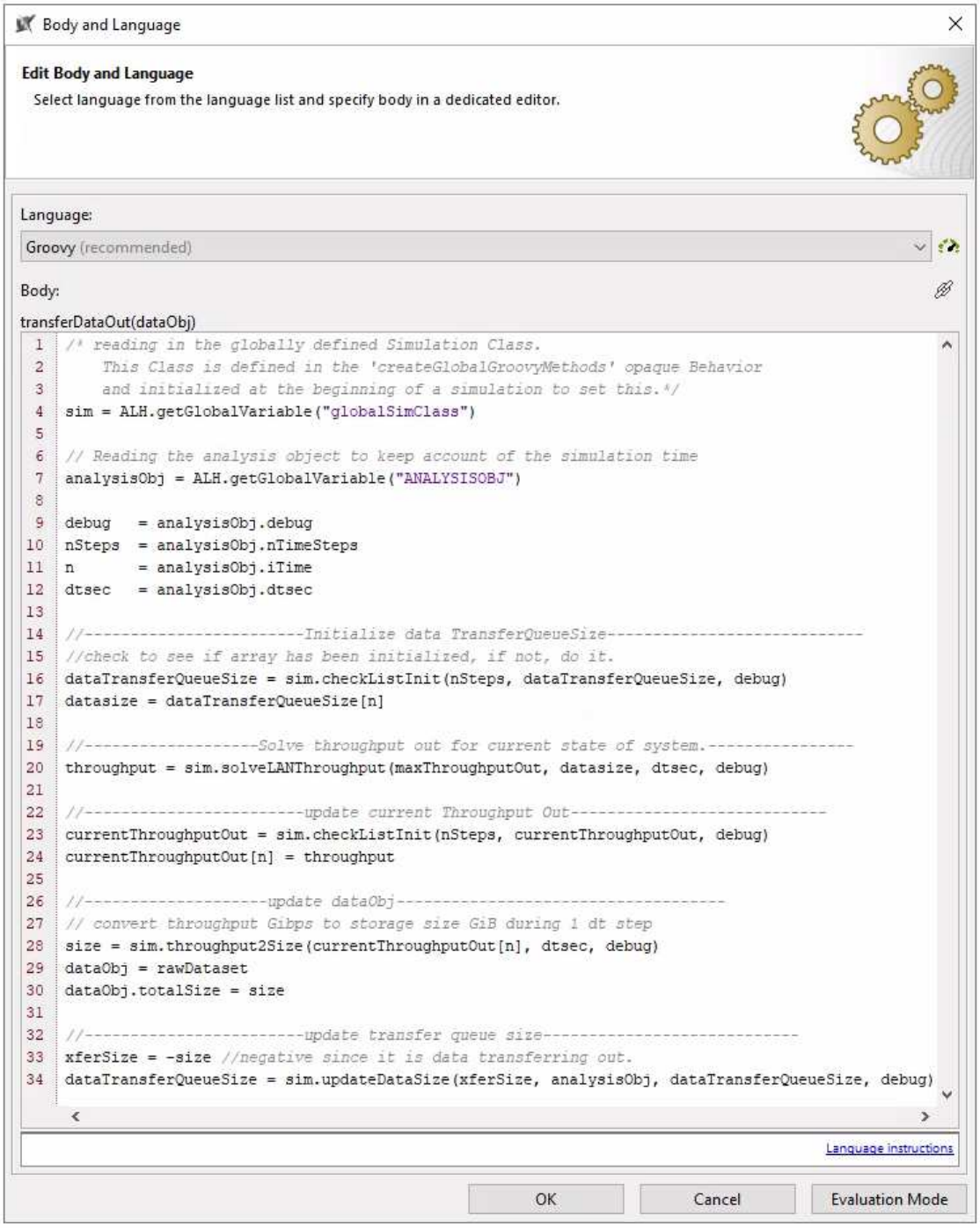


Figure 5.26: Example Opaque Behavior "*transferDataOut*" (Model Ver.2)

designed to enable future expansion of the UM. This expansion in both size and complexity is demonstrated in later sections.

5.3.1.4 MDAO: UMA Simple ETL Pipeline Simulation Scenarios

This section describes the configuration of two scenarios simulated to test the functionality of the UM simple ETL pipeline system. These scenarios demonstrated the system effects from the four decision variables chosen. Our goal was to see the effects of these decision variables on system emergent behaviors, total time to process recorder data, and cloud storage cost as a function of time. The simulation algorithm logic begins data transfers when the recorder *totalRecorderDataSize* variable is greater than zero.

Table 5.6 shows the two simulation scenarios. Scenario 1 recorder's maximum throughput out in Gigabytes per second (Gbps) is slower than the throughput out of the server which enables this scenario to upload data to the cloud as fast as the recorders throughput allows. Scenario 2 is the opposite and will have its recorder transfer queue build up while it is slowly uploaded to the cloud.

We varied the two scenarios by four decision variables that we've mapped to 'business rules' which affect cloud system performance and cost. The following four business rules describe the function they serve and the potential cost or performance effect.

Business rule 1 deletes the server data after it is uploaded to the cloud which is mapped to the *deleteXferData* decision variable.

Business rule 2 is a data pipeline orchestration tool time interval setting. This is mapped to the *schedulerDtRunInterval* decision variable.

Business rule 3 is an SDI resource allocation which limits the number of pipeline processes allowed to run in parallel. This is set by the *maxNumPipes* decision variable.

Business rule 4 is based on the decision to delete the raw recorded OLTP data stored in the cloud after processing and loading into the data store. This rule is mapped to the *deleteRawData* decision variable.

Table 5.6: ETL Pipeline System Scenarios

Value Name	Scenario 1	Scenario 2	Units
Simulation Variables			
dtmin	1.0	1.0	min.
totalAnalysisTime	60	60	min.
Specification Variables			
Recorder			
maxThroughputOut	2.6	4.0	Gbps
storageCapacity	1000	1000	Gibibytes (GiB)
totalRecordedDataSize	500	500	GiB
Server			
deleteXferData	true	false	boolean
maxThroughputIn	4.0	8.0	Gbps
maxThroughputOut	8.0	2.6	Gbps
storageCapacity	400	400	GiB
Cloud			
maxThroughputIn	10	10	Gbps
maxThroughputOut	4.0	4.0	Gbps
storageCapacity	1000	1000	GiB
storageCostRate	5.0E-7	5.0E-7	$/GiB/min.
Pipeline			
deleteRawData	true	false	boolean
singlePipeThroughput	8.66	8.66	Gbps
allowSchedulerTrigger	false	false	boolean
schedulerDtRunInterval	5.0	3.0	min.
maxNumPipes	5	2	integer
compressionRate	0.5	0.5	real

5.3.1.5 MDAO: UMA Initial Simulation Results

Figures 5.27 and 5.28 show the time-series plots of the simulation value properties from scenarios 1 and 2, respectively. Figures 5.27(a) and 5.28(a) show the data sizes (GiB) as a function of time for scenario 1 and 2, respectively. The *xfer queue* is shorthand for data transfer queue.

The server storage size and server transfer queue in Figure 5.27(a) remain near zero which verifies scenario 1 configuration worked correctly. These values correspond with the server throughput being higher than the recorder throughput and the deletion of server data after it is uploaded to the cloud (rule 1). Figure 5.28(a) shows the server storage and transfer queue gradually increase over

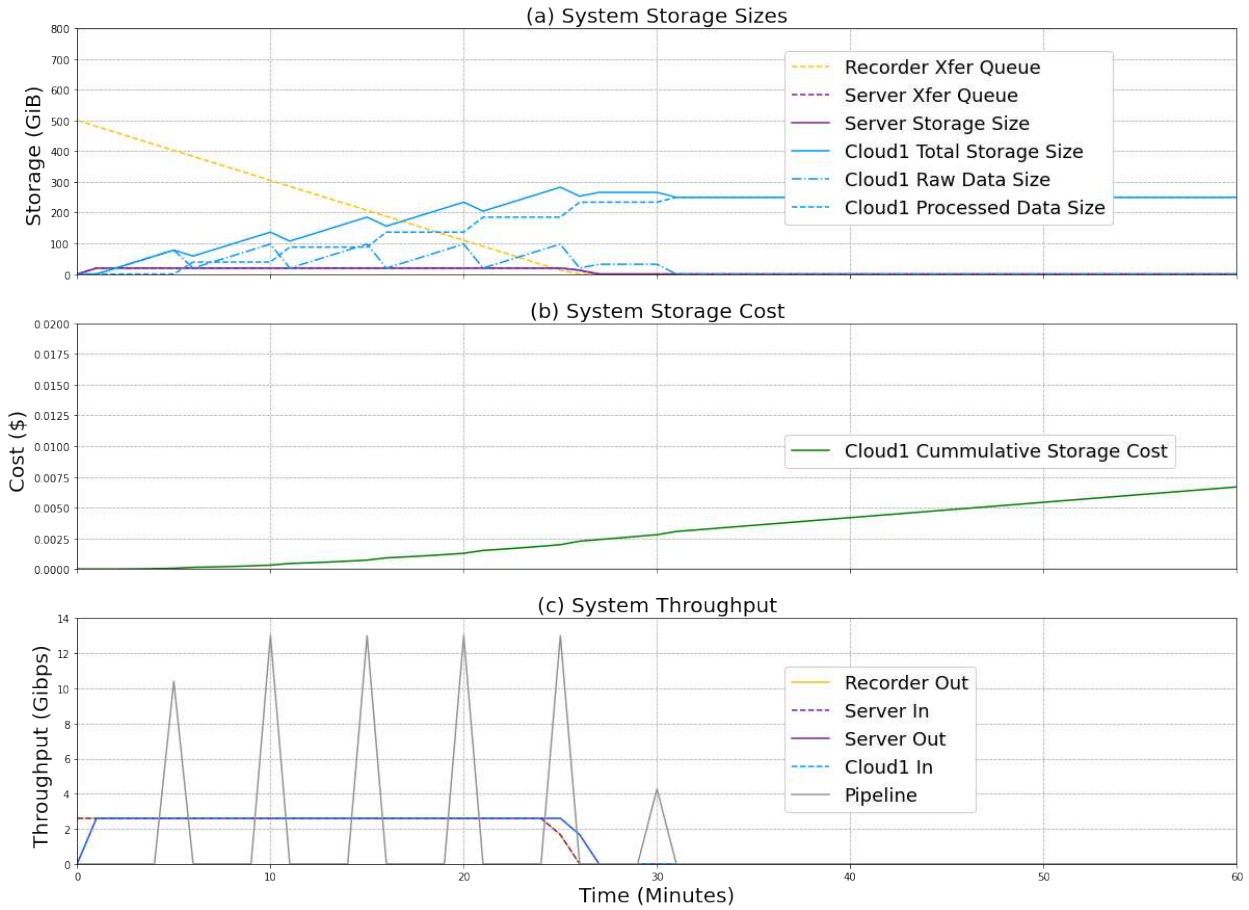


Figure 5.27: Scenario 1 Performance and Cost Results (Model Ver.2)

time as it uploads data to the cloud at a slower rate than it receives it. This verifies scenario 2 simulation configuration is working properly.

Figures 5.27(b) and 5.28(b) display the cumulative cloud storage cost as a function of time for scenarios 1 and 2, respectively. The final cost difference is the cost of storing raw data in the cloud after being processed (rule 4). Figures 5.27(c) and 5.28(c) plot the throughput for the recorder, server, cloud, and ETL pipeline for scenarios 1 and 2, respectively. These figures highlight the difference in the scenario 1 and 2 pipeline throughput spikes as a function of their different maximum parallel pipes (rule 3) and time intervals (rule 2). Scenario 2 has shorter throughput spikes that occur more frequently compared to scenario 1.

In terms of performance and cost comparisons between these two scenarios, Figures 5.27(a) and 5.28(a) show that scenario 1 took longer to finish processing the data than scenario 2. However,

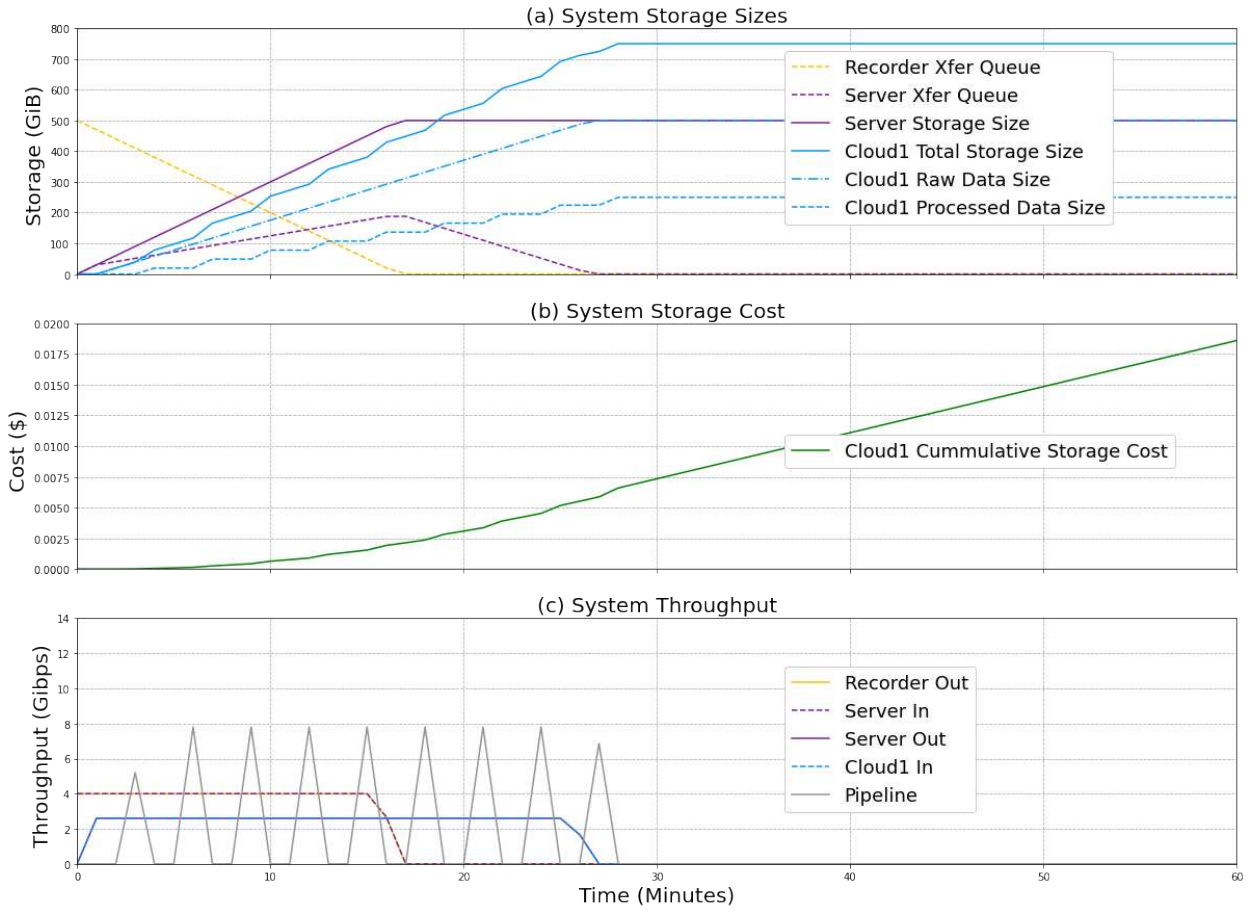


Figure 5.28: Scenario 2 Performance and Cost Results (Model Ver.2)

Figures 5.27(b) and 5.28(b) show that scenario 1 cost less than scenario 2 by the end of the 60 minute simulation.

5.3.1.6 MDAO: UMA Initial Simulation Discussion & Conclusions

This modeling and simulation effort was successful at producing results that were anticipated before running the simulations. These results were verified by simple hand calculations. These simulations demonstrated that a simple UM can capture performance, costs, and emergent system behaviors of a data system.

We concluded that these results support our assertion that the UMA adds value to MBSE for SE teams designing data systems. These UM simulations provided the SE team with performance and cost insights for the data system business rules that will affect design decisions. Furthermore,

lessons learned from this UM are applied to the next UM version (Ver.3). Many of the simulation functions and utilities are removed from this UM to create the model model libraries shown in Figure 5.4.

5.3.2 *Dev-V&V*

During this second YAMM iteration, the SW-Dev and the DI teams made significant progress on MVPs 1 & 2. They also started MVPs 3 & 4 after the vendors for the last two data formats delivered their recording and translation specifications. Figure 5.5 shows the WBS items that were started and finished in this *Dev-V&V* stage for YAMM iteration 2, however we'll also summarize the results in this section.

While the DI team was getting the operational data system ATO and network connections approved they were able to deploy the DevSecOps environments. This is shown by WBS items DI 1.2 and DI 1.3 as shown in Figure 5.5. The DevSecOps environments were built and deployed in AWS as shown in the logical diagram in Figure 5.10. The deployment of the DevSecOps environments allowed the SW-Dev team to finish development of the MVP 2 cloud native Data Product as shown by WBS SW 2.2 on the schedule in Figure 5.5.

Additional SW-Dev teams were added to develop the ETL pipelines for data formats 2 & 3 in parallel with MVPs 1& 2 efforts. This new work was defined by MVPs 3 & 4 shown in the WBS schedule in Figure 5.5. After the vendors of these formats delivered their ICDs, the new SW-Dev teams were able to design and develop the new ETL pipelines in this YAMM increment. These pipelines were integrated with the current Data Product developed in MVP 2. All SW-Dev teams provided feedback to the DI team about defects or lack of resources in the SDI. This iterative feedback and development between the SW-Dev teams and the DI team ensured the cloud native Data Product was able to scale up to meet the demands of the three ETL data pipelines.

5.3.3 *Operational Data System*

The approvals of the ATO, network connections, SDI software, and MVP 2 Data Product software happened during this YAMM iteration. After these approvals, the DI team was able to deploy

the initial version of the operational data system which is represented in Figure 5.14. The local server and Daqscribe Wide Area Networks (WANs) had not been connected yet so team utilized AWS Snowballs to upload OLTP aircraft data to the operational data system staging bucket as a temporary solution.

After the initial operational cloud environment was ready the SW-Dev team deployed the initial MVP 2 Data Product. At this point, the OLTP data was being uploaded into the operational staging bucket, therefore the ETL pipeline was able to start execution. A few initial end users began using the Data Product and also began providing feedback to the SW-Dev team.

Aircraft data was continually loaded into the operational staging bucket using AWS Snowballs for most of this YAMM iteration. Meanwhile, AWS collected the resource data and system metadata from the DevSecOps cloud environment and the operational cloud environment.

5.3.4 *Data Analytics & Algorithmics*

This is the first YAMM iteration which collected statistical data from the initial DevSecOps and operational environments. However, most of this data was reported by the SW-Dev or DI teams to the SE team rather than through empirical means and data collection efforts. The data transfer rate from the on-prem "local" server shown in Figure 5.14 was provided by the DI team and their method of measurement was from a couple of files transferred. This method was not based in statistics, rather it was an observation that was helpful to the SE team in its preliminary stages of a simulation-enabled model. The data transfer rate provided from the on-prem server across the WAN was 9 Gbps. This measurement turns out to be inaccurate in future data analysis measurements, but that is discussed in later sections.

Initial process throughput for a single ETL pipeline for the MVP 2 Data Product was estimated by the SW-Dev team using. The parallel nature of running these ETL processes made this difficult, but they estimated 0.139 Gbps for each process. The current resource limit at the time limited the ETL processes to 20 running in parallel. This was also difficult to measure since there were pipeline orchestration processes, K8s overhead, and other overhead processes taking up part of

the 20 node limit. There were also limits to the input and output of the AWS VMs, the S3, and the transfer between different AWS accounts. The different staging bucket account and the Data Product accounts are shown in Figure 5.14.

These initial estimates were fed back to the SE team to incorporate into their initial simulation-enabled models. In addition to these estimates, SE team were updating the Ver.2 simulation-enabled model in parallel to increase its fidelity to better represent a full Data Product rather than a simple ETL pipeline as shown in Figure 5.19. The results of the SE team's empirical feedback and simulation-enabled model updates are discussed in the next section.

5.4 YAMM Iteration 3

In this section, we review the work done during the third YAMM iteration for the hybrid cloud case study. In this iteration all MVPs were deployed to the operational data system and were being continuously updated, tested and re-deployed based on user feedback and defects. Without new MVPs generated during this iteration we moved into the continuous YAMM analysis and optimization process laid out by the YAMM Process Flow 1 in step 5.1.

Within this continuous YAMM process, the aircraft OLTP system aggregated the last of its data for this case study into the operational data system. The CSP data collection software was continuously collecting resource usage and cost data about the software deployments, data aggregation, and the execution of the ETL pipelines. Finally, the resource data was analyzed and fed back to the SE team where they updated the UM and ran initial system optimizations. The rest of this section describes the results of these activities within each YAMM stage.

5.4.1 *MDAO*

This section describes the Ver.3 model and the new capabilities the SE team created. The SE team made this version more modular than Ver.2 by creating and using reusable libraries as shown in Figure 5.3. These libraries were based on the simulation components that were split out from the

Ver.2 model. The Ver.3 model also used the specification libraries created with the Ver.1 model. This usage is also shown in Figure 5.3.

In terms of capabilities added to the Ver.3 model, the SE team increased the number of components compared to the ETL data system, added storage and cost rollups, enhanced the Simulation Viewpoint visibility, and updated the simulation to be called from a python environment. The updates to the UM Ver.3 are discussed in more details in Sections 5.4.1.1 through 5.4.1.3, which are labeled as each UMA metamodel layer.

The ability to call the Ver.3 model from a python environment enabled us to execute a system optimization algorithm on this model. Objective functions for cost and performance were created and a PSO was built to optimize these functions over a set of weighted coefficients. The discussion of the simulation parameters, specifications, and the Pareto-optimal results are in 5.4.1.4.

5.4.1.1 MDAO: UMA Metamodel Visual Specification Layer

In this section, we describe the new capabilities the SE team created in the Ver.3 UM in the visual specification layer. We also describe this model's new dependencies on model libraries that are in the lower two simulation layers in the UMA metamodel as shown Figure 3.4.

We start this discussion with Figure 5.29, which is the updated definition of the Abstract Data Sub-System. This subsystem element has the value properties, proxy ports, reference properties, and constraints necessary for the structural side of a simulation-enabled model to execute.

The block in Figure 5.29 is an upgrade from the Ver.2 version shown in Figure 5.17 and now resides in the Resource and Simulation Library shown in Figure 5.3. The upgrades to this block include value property roll-up constraints for throughput and storage, time series arrays for storing the history of the state of the simulation values, and reference properties that enable each simulation component to pass data objects via activity diagram object flows to the next component.

Figure 5.30 shows elements from the Data Mesh Library and Resource & Simulation Library that inherit the properties from the Abstract Data Sub-System block. This figure also shows two of the new stereotypes, *Data Product* and *Data Domain*, used from one of our model profile libraries.

Figure 5.29: Abstract Data Sub-System Definition (Model Ver.3)

This figure also shows cloud-based versions of the Data Product and Data Domain with multiple inheritance to enable cloud cost calculation capabilities.

Figure 5.31 shows the Ver.3 hybrid cloud Data Mesh system composed of elements inherited from the Data Mesh Library and the Resource & Simulation Library. Using these libraries enables a new data system to be modeled quickly using the UMA. This figure also shows the subsetted part definitions of each data system part which enables storage and cost rollups. These rollups are for simulation purposes and will be discussed in the next section.

Figure 5.32 shows the composition of the Data Domain block. The Cloud Based Data Domain in the data system in Figure 5.32 inherits properties from this block and the Abstract Cloud block

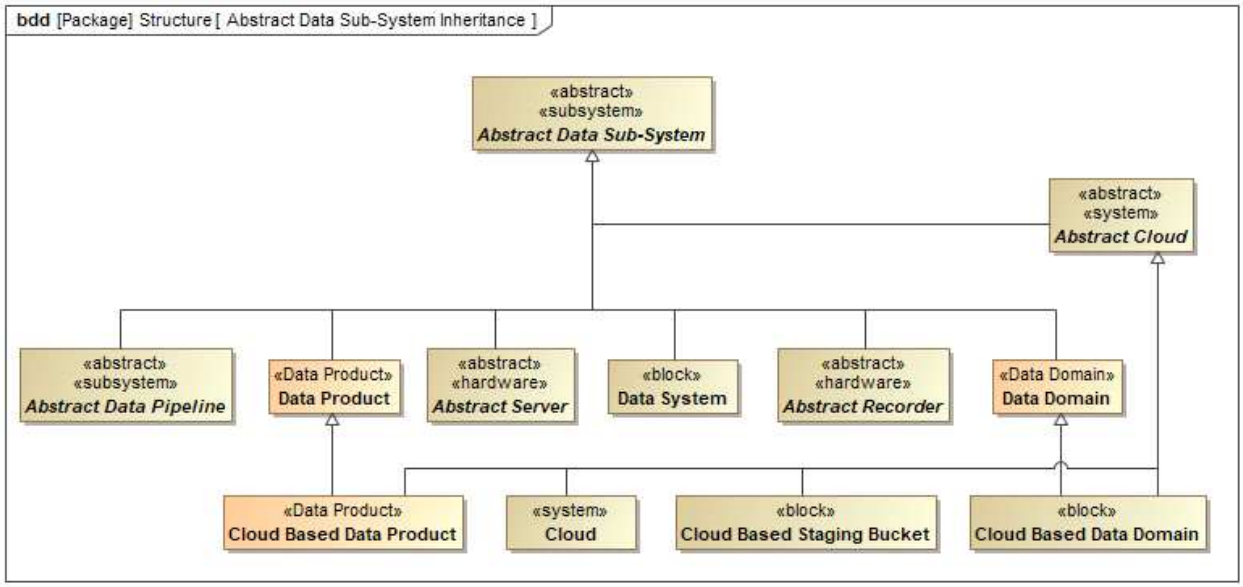


Figure 5.30: Abstract Data Sub-System Inheritance (Model Ver.3)

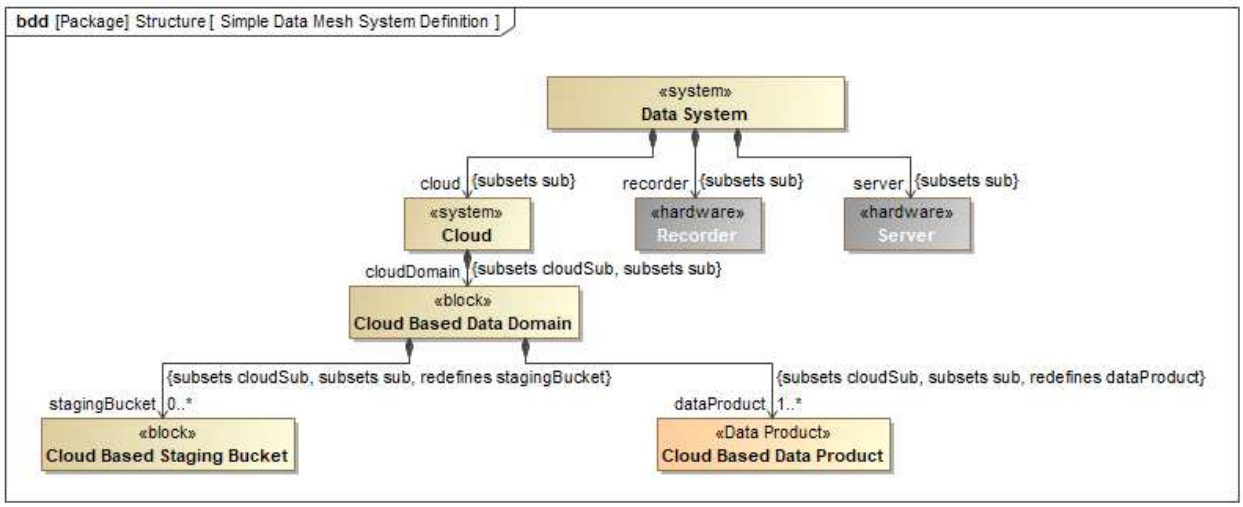


Figure 5.31: Simple Data Mesh System Definition (Model Ver.3)

as shown in Figure 5.30. Figure 5.32 shows the subsetted part definitions that enable storage roll-up constraints to be applied. This figure also highlights the additional stereotypes inherited from the profile libraries shown in Figure 5.3 and the settings applied in the tool to easily separate stereotypes by color.

Figure 5.33 shows the internal structure of the data system and the data flows between subsystem parts. This figure shows the increase in Ver.3 model fidelity by the addition of a cloud domain,

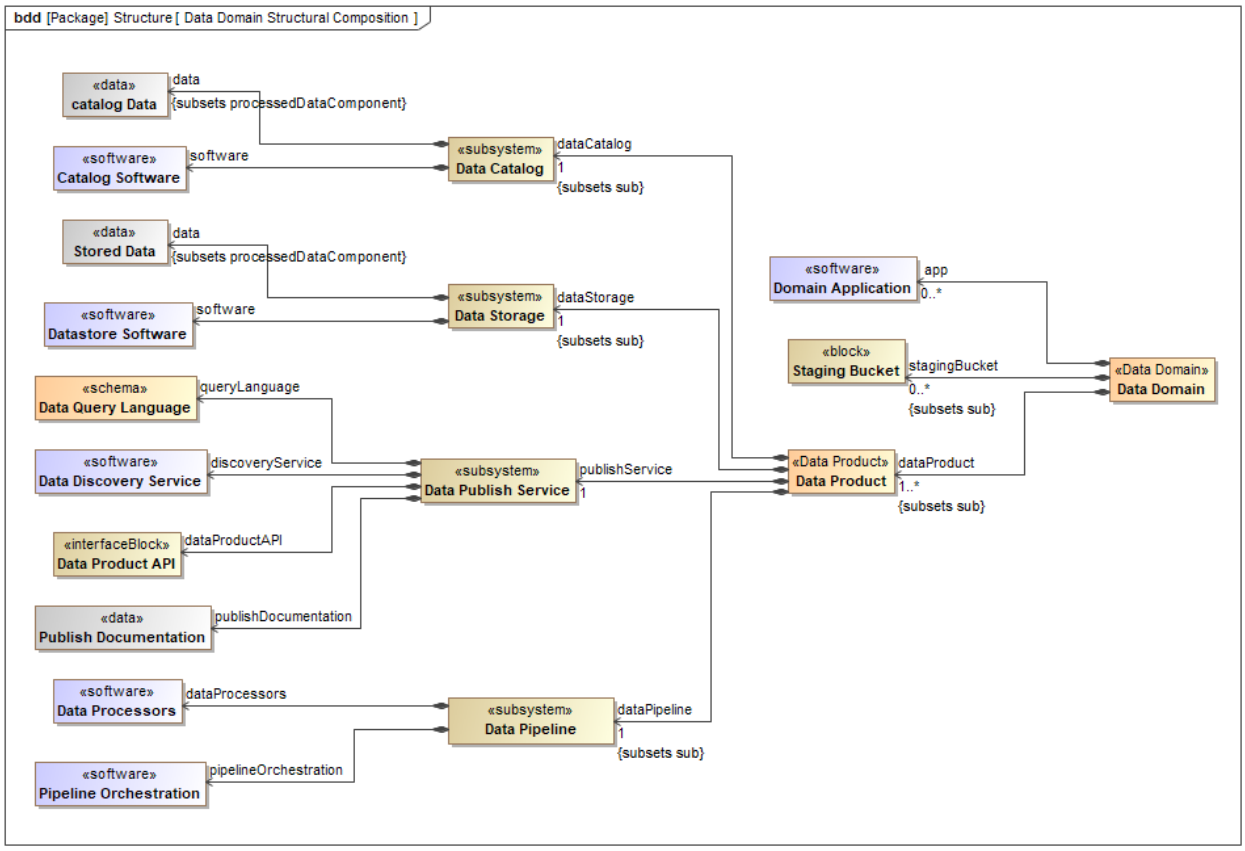


Figure 5.32: Data Domain Composition (Model Ver.3)

Data Product, and a staging bucket, compared to the Ver.2 simplistic model shown in Figure 5.19 without these components. This model represents an incremental step towards representing the specifications shown in the updated Ver.1 model in Figure 5.14 where the staging bucket is outside of the Data Product which is composed of a data pipeline, data catalog, datastore, and data publish service.

The increased fidelity of the structural components necessitates an update to the behavioral components for the simulation to work. Figure 5.34 shows the primary use case of this hybrid cloud case study and its activities allocated to the data system structural elements. This figure is similar in structure to the Ver.2 model simple ETL behavior diagram in Figure 5.20 and reuses the first three activities from this model. The upgrade to the system behavior is encapsulated in the *Update Data Product from Staging Area* activity.

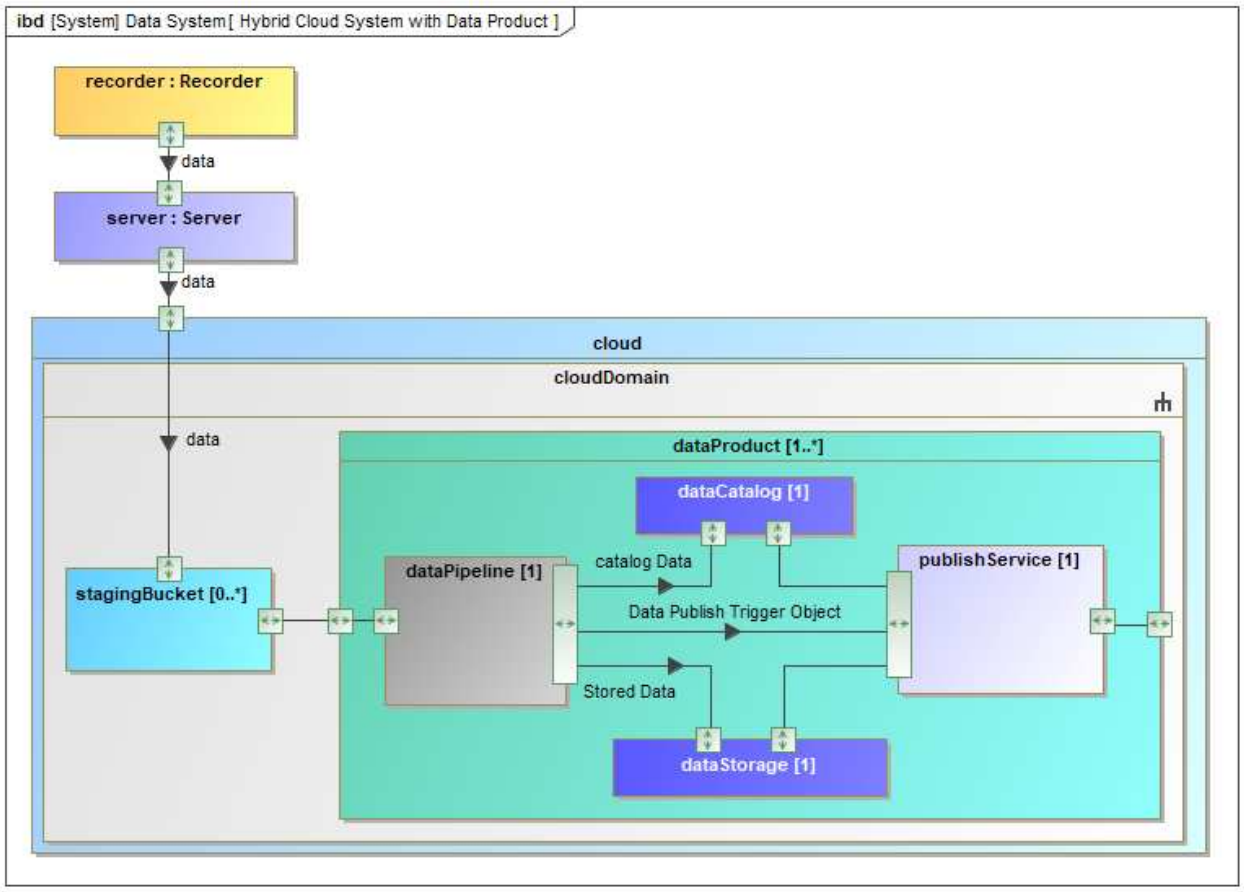


Figure 5.33: Hybrid Cloud Data System with Data Product (Model Ver.3)

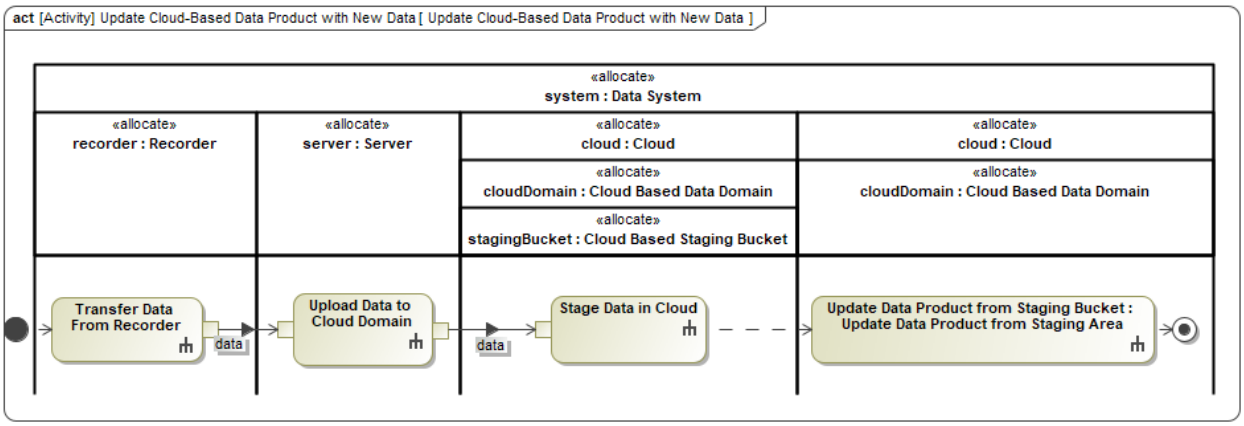


Figure 5.34: Hybrid Cloud Data System Primary Use Case (Model Ver.3)

Figure 5.35 shows the upgraded specification behavior built by the SE team. This behavior diagram informs the SW-Dev team when developing a cloud based Data Product for this data system. Logic specifying key Data Product behavior is built into this diagram to direct the SW-Dev

team in their subsystem design. Two key decision nodes are present in this figure which are derived from business rules we discussed for different cloud scenarios in Section 5.3.1.4. The first decision is to determine when to run the data pipeline based on the staging bucket data processing queue. The second decision is to determine if raw data from upstream subsystem should be deleted after processing. We built these decision nodes in order to utilize future optimization decision variables towards reducing cost and increasing performance.

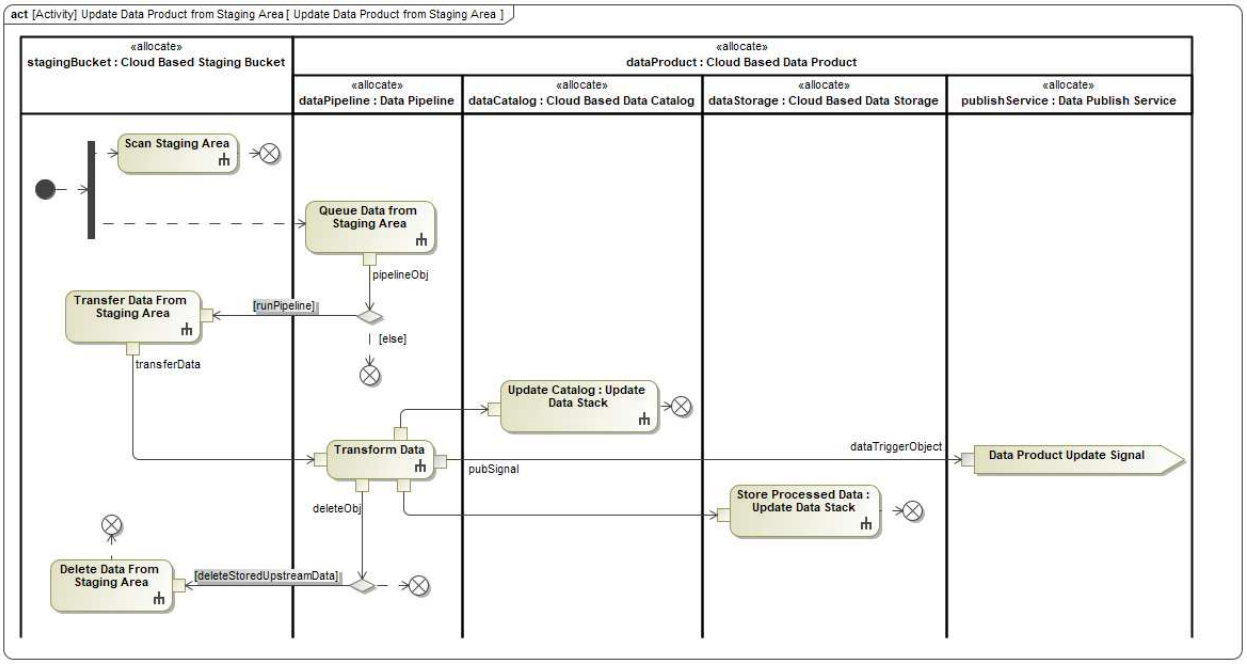


Figure 5.35: Cloud Based Data Product Behavior (Model Ver.3)

5.4.1.2 MDAO: UMA Metamodel Visual Simulation Layer

We discuss the visual simulation layer additional capabilities the SE team added to the Ver.3 model compared to the Ver.2 model. For this model, this layer of the UMA metamodel contains the most structural and behavioral elements and diagrams compared to the other two layers. This is the layer that connects the specifications to the simulation functions and utilities to enable modularity and ease of reuse.

We upgraded the Ver.3 model to run via server-side simulation in Teamwork Cloud (TWC) from a python environment. This required removing many of the ALH calls. It also required the addition of the “History” variable arrays to return time series data to the python environment.

The majority of the capabilities we added to the Data Mesh Library, Resource & Simulation Library, and the Data Mesh Profiles library are simulation elements and behaviors compared to the few specification capabilities discussed in Section 5.4.1.1. We start this discussion with the additional simulation capabilities added to the Abstract Data Sub-System block shown in Figure 5.29. Figure 5.36 shows the *sub* part property of the Abstract Data Sub-System block and its status as a derived union. A derived union is defined as a type of property that represents the combined set of values from other properties that are designated as its "subsets". properties that are constrained to subset it. This type of property enables data and cost value rollups from any number of children, or children of children, to a final parent block.

The *processedDataComponent* and *upstreamDataComponent* in Figure 5.36 are also *Derived Union* properties. This enables the data catalog and datastore, which contain these data components, to roll up the storage values. The parametric diagram in this figure contains the constraints that sum all children storage values and add them to the parent’s storage value. We used three constraint blocks to roll up the upstream data stored in the system, the processed data in the system, and the total amount of data in the system. We separate these to better understand how much raw upstream data stored in the system contributes to the cloud costs we are trying to reduce.

While all system blocks are specializations of the Abstract Data Sub-System block, only the cloud-based blocks inherit attribute from the Abstract Cloud block too. SysML allows inheritance from multiple classes and we use this to add cloud costs to regular sub-system blocks. This block’s definition is shown in Figure 5.37 along with its storage cost rollup constraint and *Derived Union* part property. This enables all child cloud storage blocks to sum storage costs up to the system level. This provides information for the system optimization objective function calculations.

Figure 5.38 shows the abstract Simulation block definition and its constraints. These constraints calculate the total analysis time and convert minutes to seconds. The block keeps track of

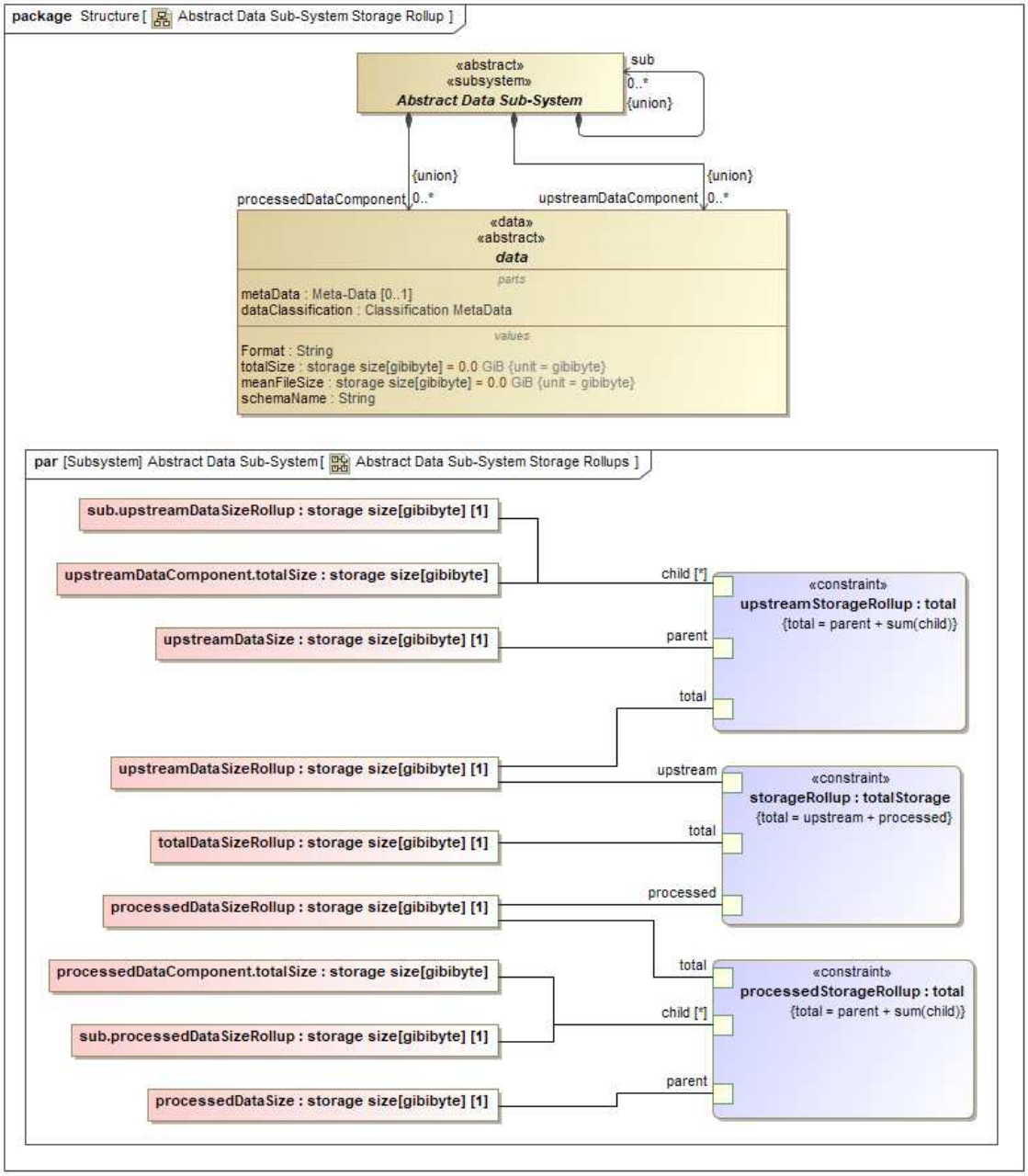


Figure 5.36: Abstract Data Sub-System Constraints and Storage Rollup (Model Ver.3)

the current time step index and provides these simulation variables to all simulation functions and utilities like a set of global variables. All simulation blocks, which simulate system and subsystem use cases, inherit its definitions. The *Update Cloud-Based DP Simulation* block defines the simulation for the primary use case of this hybrid cloud case study, which is described in Section 3.1.1.3 and is represented as the "Sim Blocks" in Figures 3.5 and 3.6.

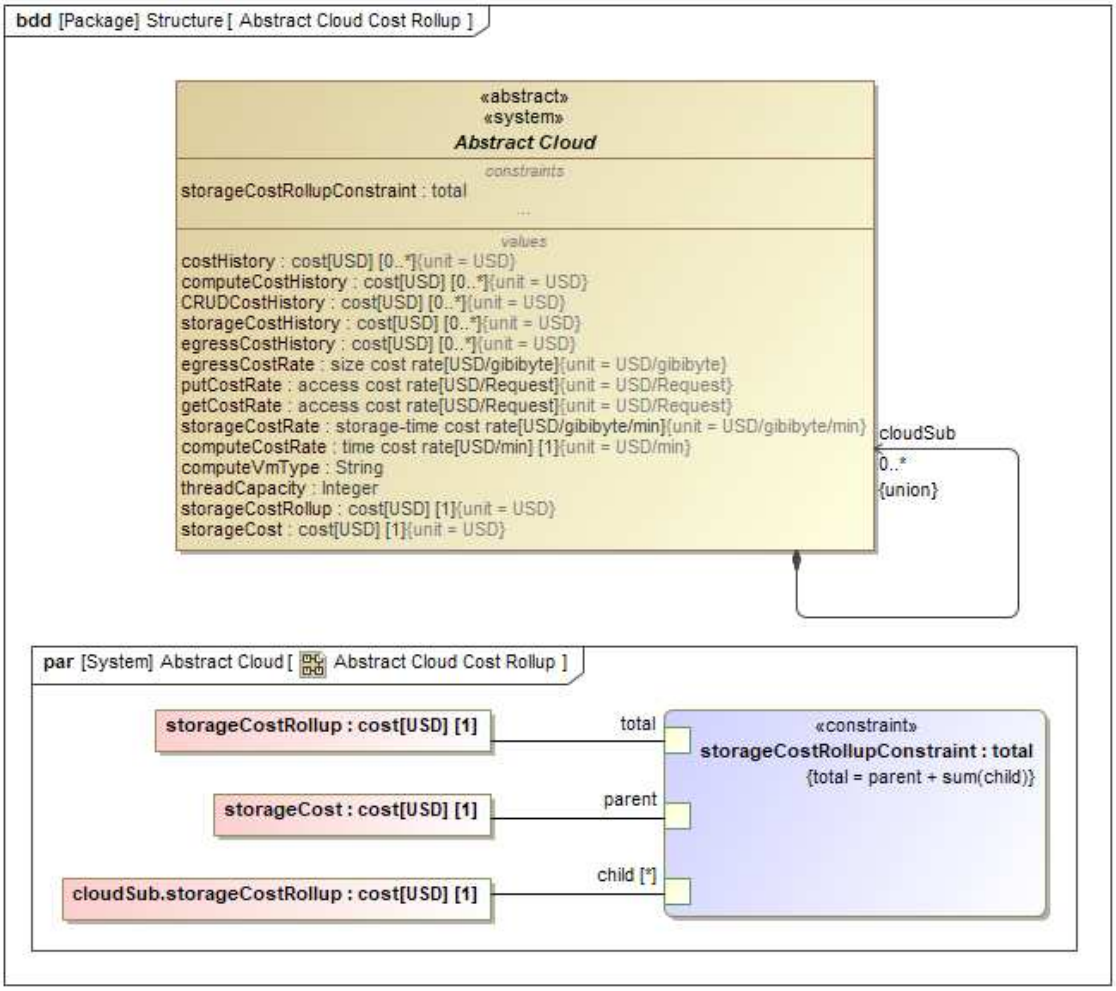


Figure 5.37: Abstract Cloud Constraints and Cost Rollup (Model Ver.3)

Figure 5.38 also shows the new *Simulation Block* stereotype meant to better enable model users to the quickly distinguish UM Simulation Viewpoint from a Specification Viewpoint. These viewpoints are defined by the UMA metamodel layers as shown in Figure 3.4. We changed the DS cameo systems modeler (csm) tool settings to color all Simulation Blocks blue with a dark blue border for users to distinguish from specification blocks. Figure 5.39 shows the use of the *Simulation Behavior* stereotype which sets the blue outline of activity diagrams for simulation behaviors.

The system use case being simulated in Figure 5.39 is the primary use case defined in Figure 5.34. Figure 5.39 shows the simulation behavior of the simulation block (*Update Cloud-Based DP Simulation*) defined in Figure 5.38. This simulation behavior contains opaque behaviors that

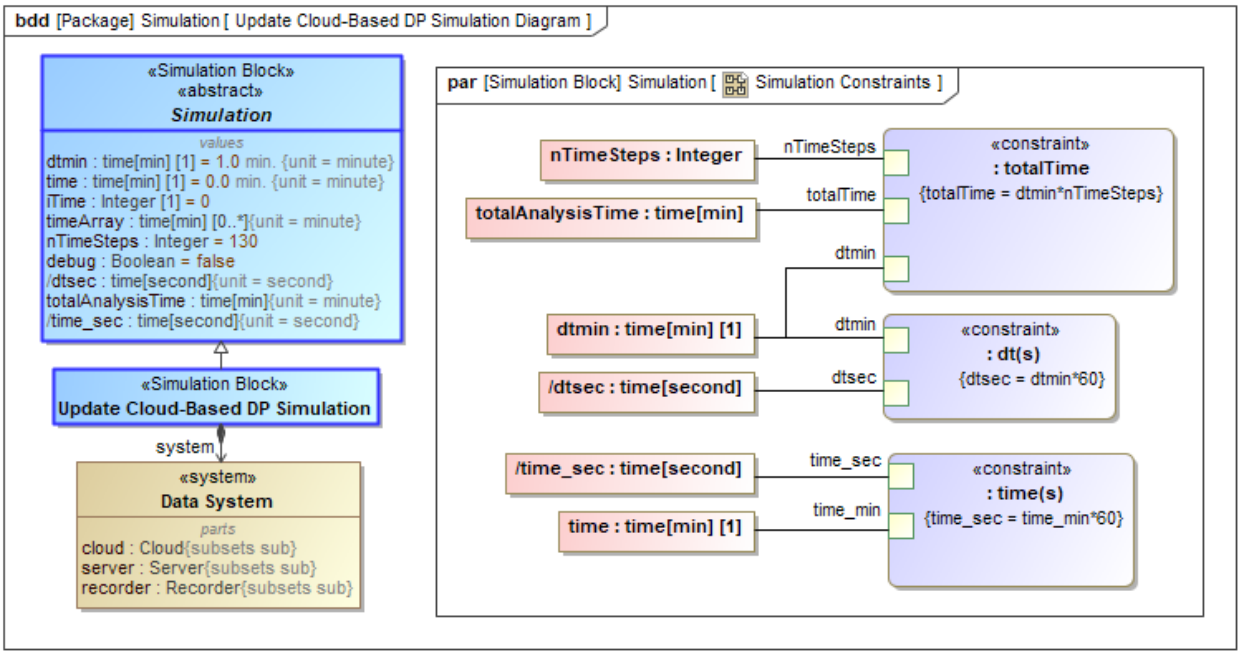


Figure 5.38: Abstract Simulation Definition & Constraints (Model Ver.3)

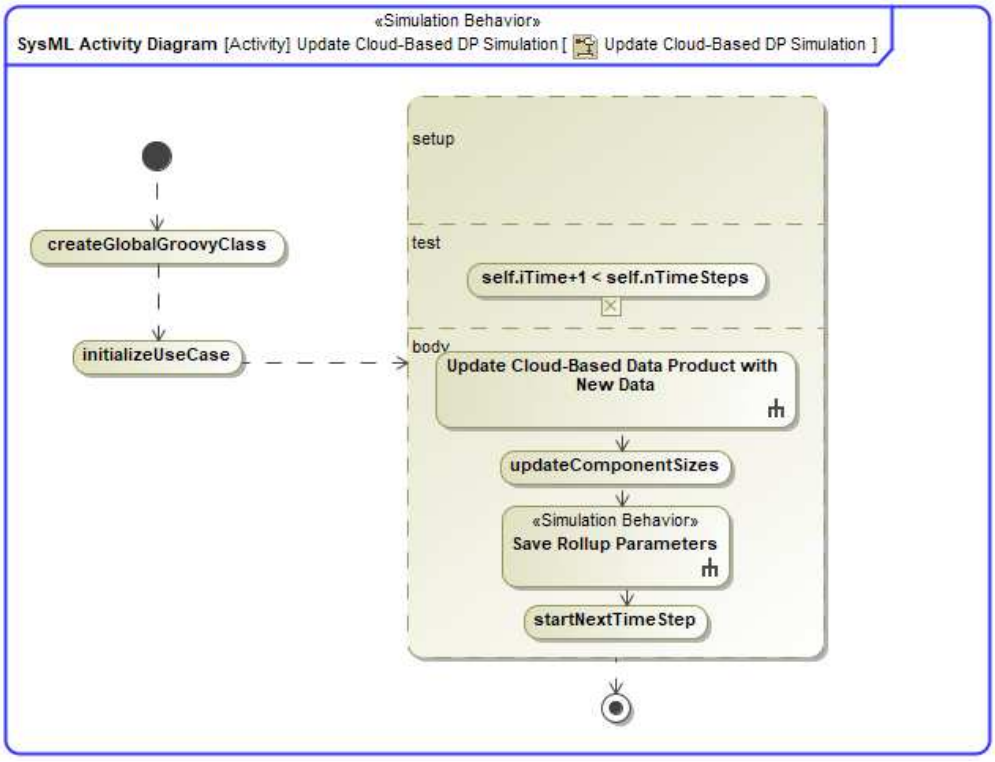


Figure 5.39: Primary Use Case Simulation Behavior (Model Ver.3)

create the global Groovy Simulation class, initialize the primary use case, simulate the primary use case, save storage rollup values, and increment the simulation time.

Figures 5.40, 5.41, and 5.42 are the visual simulation behaviors for the first three specification behaviors in the primary use case shown in Figure 5.34. Figure 5.40 shows a simulation behavior which only contains a single opaque behavior. While this appears redundant this strategy enables future simulation capabilities to be added to this simulation behavior without the need to modify the use case specification. This strategy also enables reuse of our modular behaviors such as the *transferDataOut* opaque behavior in this figure and Figure 5.41.

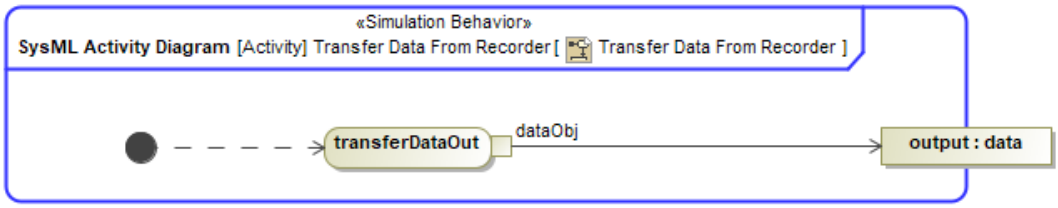


Figure 5.40: Simulation Behavior *Transfer Data From Recorder* (Model Ver.3)

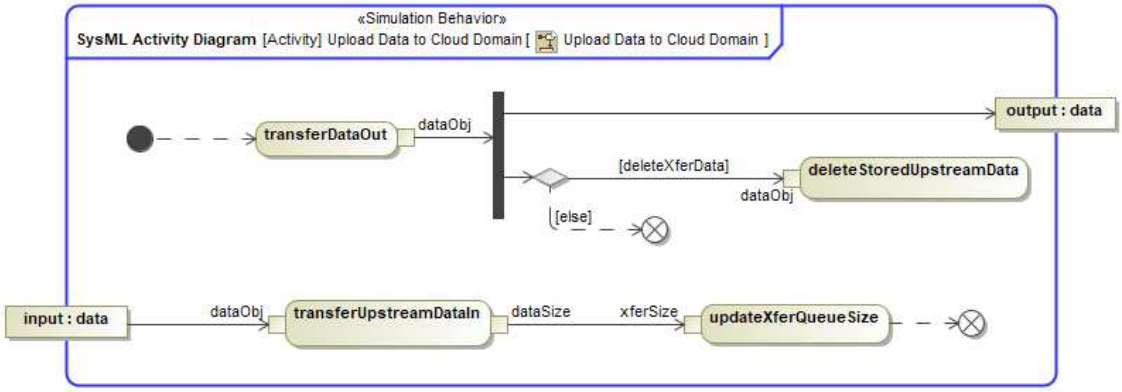


Figure 5.41: Simulation Behavior *Upload Data to Cloud Domain* (Model Ver.3)

Figures 5.43 through 5.48 are the visual simulation behaviors for the *Update Data Product from Staging Area* behaviors in the activity diagram shown in Figure 5.35. Most of these opaque behaviors simply add or subtract data from queues, or report the size of data queues. However, the *analyzeUpstreamDataForProcessing* and *processData* opaque behaviors in Figures 5.44 and 5.46

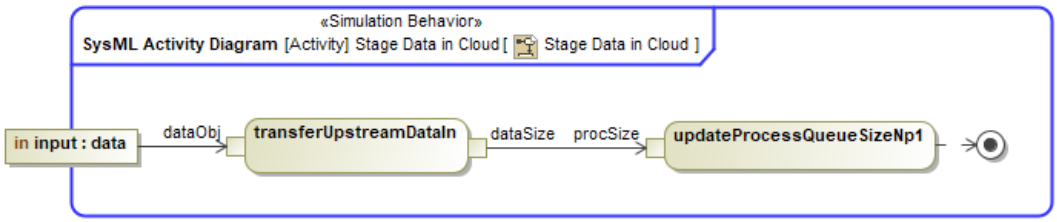


Figure 5.42: Simulation Behavior *Stage Data in Cloud* (Model Ver.3)

are more complex and contain the logic and algorithms to calculate resource usage and cost as a function of time for the ETL pipeline. These two opaque behaviors are the focus of much of the empirically derived data that helps estimate cost and resource usage.

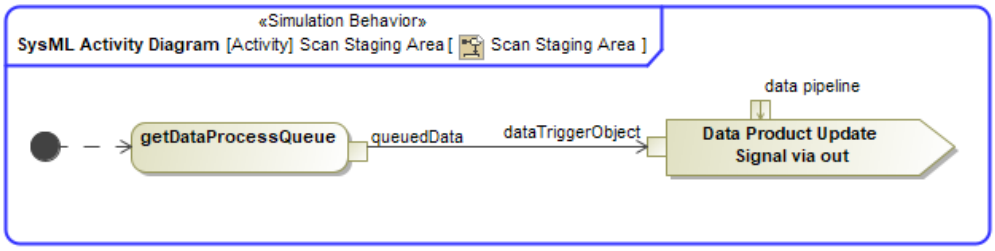


Figure 5.43: Simulation Behavior *Scan Staging Area* (Model Ver.3)

The *analyzeUpstreamDataForProcessing* opaque behavior in Figure 5.44 calculates if the data pipeline will run during the time step and how much data it will process that time step. This algorithm requires the data pipeline variables such as *singlePipeThroughput*, *allowSchedulerTrigger*, *schedulerDtRunInterval*, and *maxNumPipes*. These data pipeline variables are based on empirical data or data coming from the downstream design. The variables used as optimization decision variables to analyze their effects on cost and performance.

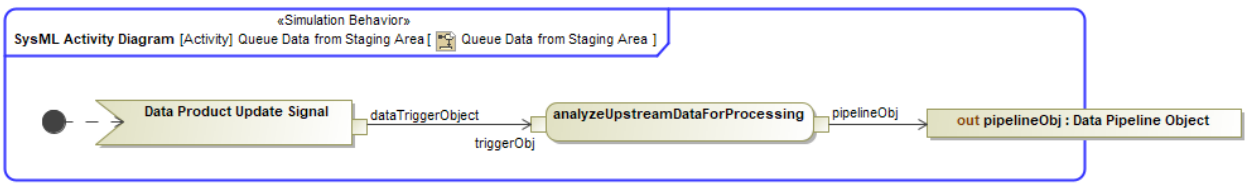


Figure 5.44: Simulation Behavior *Queue Data from Staging Area* (Model Ver.3)

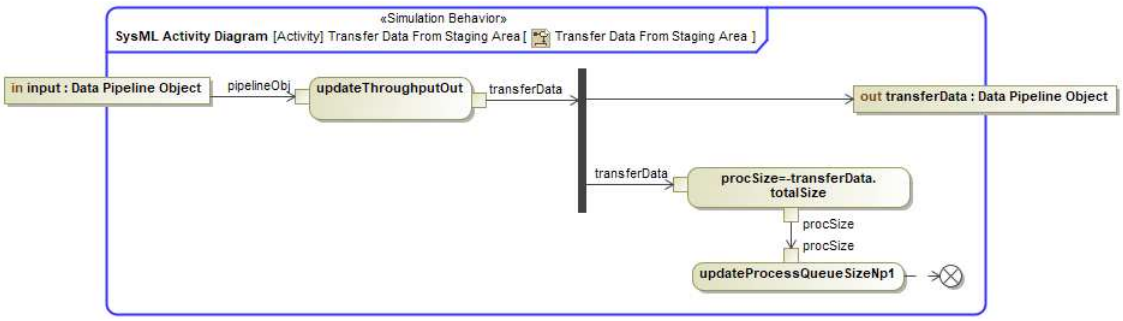


Figure 5.45: Simulation Behavior *Transfer Data from Staging Area* (Model Ver.3)

The *processData* opaque behavior in Figure 5.46 calculates the amount of processed data that will be loaded into the data catalog and the datastore separately. This calculation is based entirely on empirical data. The pipeline variables we used for this algorithm are *compressionRate*, *catalogProportion*, *datastoreProportion*, and the amount of raw upstream data that was processed during the time step.

The compression rate depends on the raw data format, recording format, the data type being loaded into the catalog and datastore, and the software that manages the data catalog and datastore. These are some of the latent variables we figured out during the course of this case study. The AA team provide a statistical mean for all of these values during the last YAMM iteration. The *processData* opaque behavior also calculates the size of raw data to delete from the staging area based on the data it processed. However, if the boolean *deleteStoredUpstreamData* is false, then this data will not be deleted. This boolean is another optimization decision variable built into the model by the SE team.

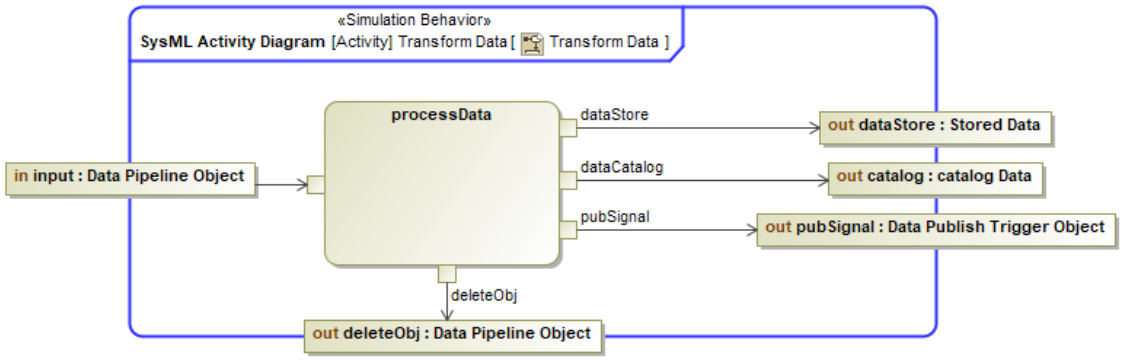


Figure 5.46: Simulation Behavior *Transform Data* (Model Ver.3)

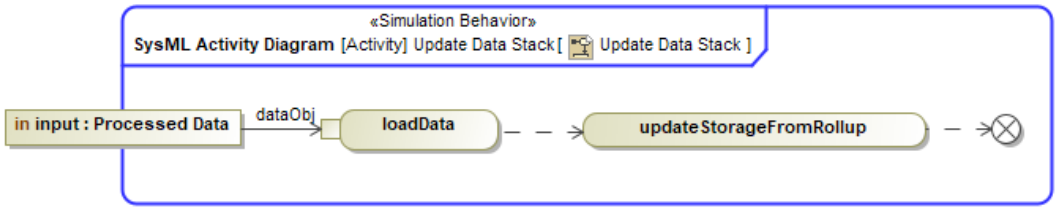


Figure 5.47: Simulation Behavior *Update Data Stack* (Model Ver.3)

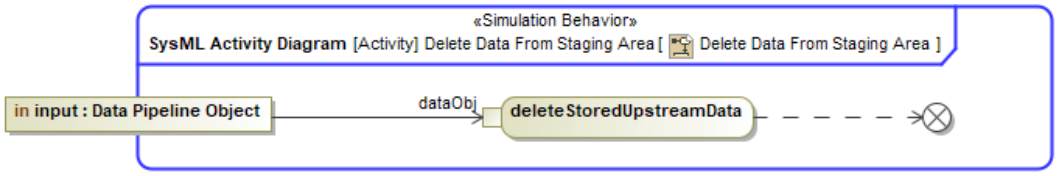


Figure 5.48: Simulation Behavior *Delete Data from Staging Area* (Model Ver.3)

As the SE team performs the UMA, it is vital they identify the key decision variables in the model to be able to take advantage of them during system optimization. Additionally, they must ensure they can use the decision variables as inputs into the model and know the range of each variable.

5.4.1.3 MDAO: UMA Metamodel Functions & Utilities Layer

In this section, we provide an overview of the Ver.3 hybrid cloud metamodel, which contains all three layers defined in the UMA metamodel structure as discussed in Section 3.1.1.2 and illustrated in Figure 3.4. Figure 5.49 is the metamodel for the Ver.3 hybrid cloud data system UM which is limited to the primary use case, its simulation block, and all of their dependencies. This figure is divided into the three UMA metamodel layers which are labeled appropriately.

The visual simulation layer in Figure 5.49 contains different element types. Activity elements sit near the top of this layer. These activity elements are the Simulation Behavior stereotypes with activity diagrams discussed and shown in 5.4.1.2. The opaque behavior elements near the bottom of the visual simulation layer are the behaviors contained in the Simulation Behavior activity dia-

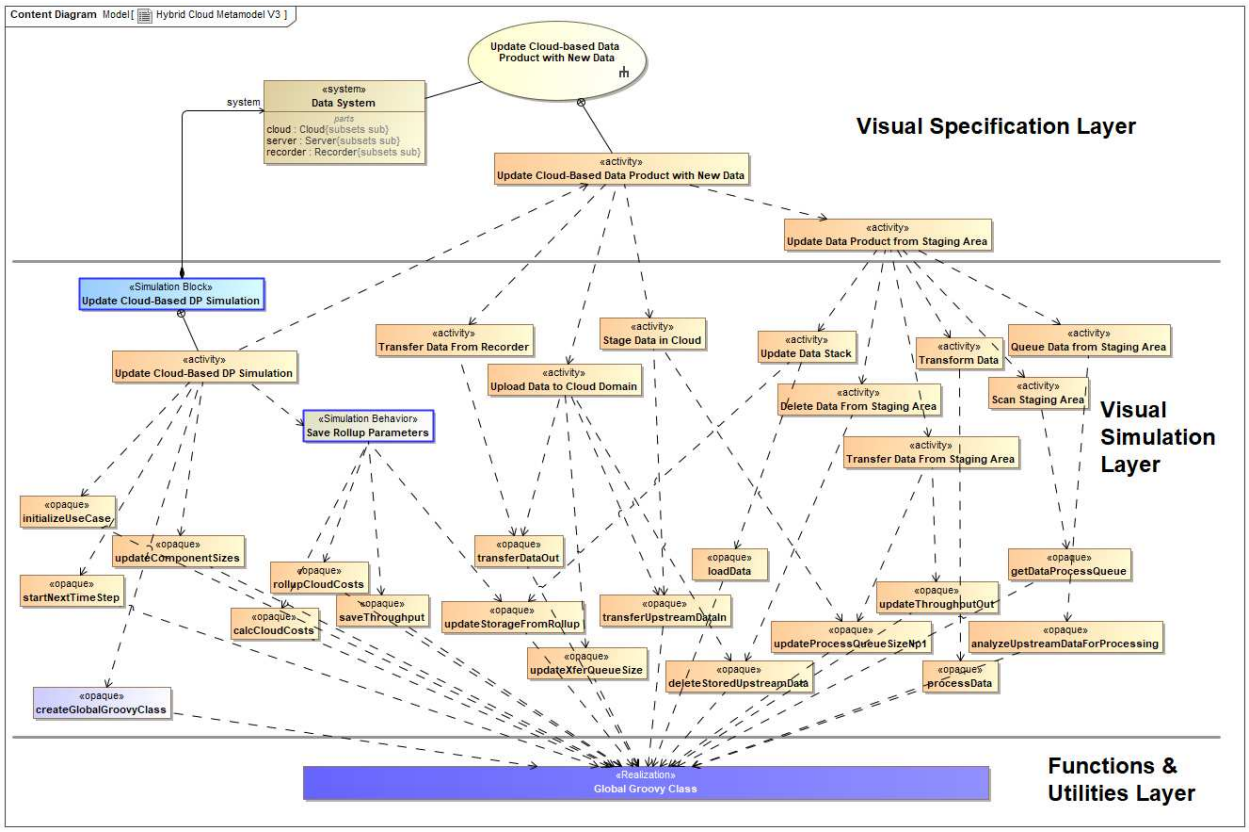


Figure 5.49: Hybrid Cloud Metamodel (Model Ver.3)

grams. These opaque behaviors contain Groovy code that call the Global Groovy Class methods in the Functions and Utilities layer.

5.4.1.4 MDAO: System Optimization

In this section, we present an overview of system optimization performed on the Ver.3 UM. This overview includes the performance and cost objective function, the decision variables, the optimization algorithm, the optimization results, and the Pareto analysis.

The objective function that we used is a function of the cost and performance objective functions so we could use a single objective PSO algorithm to find the minimum. We calculated the objective function using (5.1), where C_{cost} and C_{perf} are the weighted coefficients for the cost and performance values. WAN_{cost} and S_{cost} are the normalized internet and cloud storage costs. ETL_{time} is the normalized time it takes to transfer all data from the recorder, through the server,

up to the cloud, then processed and loaded into the Data Product. Minimizing this time would maximize our performance for this case study.

$$f(cost, performance) = C_{cost}(WAN_{cost} + S_{cost}) + C_{perf}(ETL_{time}) \tag{5.1}$$

The UM returned the time series data of the WAN cost, storage cost, and the ETL processing queue size for each simulation. The normalized costs and timing were calculated from these values. The cost and performance variables were normalized using min-max normalization to ensure all values were in the range of 0 - 1 to avoid numerical bias by large numbers.

Table 5.7 shows the four decision variables and their domains we used during the system optimization. The PSO algorithm we setup was allowed to vary the *Internet Throughput* decision variable for any real number between 0.75 and 100 Gbps, however the cost of the internet was determined by three discrete internet packages with maximum throughput of [1, 10, 100] Gbps. We did this to test if the PSO algorithm would converge on the highest throughput within a specific internet package range.

The *Delete Raw Data* boolean in Table 5.7 sets the *deleteStoredUpstreamData* variable shown on the decision node in Figure 5.35. The *Max Num Pipelines* decision variable sets the *maxNum Pipes* simulation variable the opaque behavior in Figure 5.43 that determines how many parallel processes can run in the ETL pipeline. The *Pipeline Interval* also affects the opaque behavior in Figure 5.43 and sets the *schedulerDtRunInterval* simulation variable that informs the ETL pipeline how much time it must wait in between execution.

Table 5.7: System Optimization Decision Variables (Model Ver.3)

Decision Variable	Domain	Units
Internet Throughput	$\{x \in \mathbb{R} \mid 0.75 \leq x \leq 100\}$	Gbps
Delete Raw Data	$\{false, true\}$	-
Max Num Pipelines	$\{x \in \mathbb{N} \mid 10 \leq x \leq 1,000\}$	-
Pipeline Interval	$\{x \in \mathbb{N} \mid 1 \leq x \leq 10\}$	Min.

During the system optimization, we ran our PSO algorithm using the objective function value calculated using (5.1) and inputting the cost and performance coefficients provided for that simulation. We set the PSO algorithm number of particles to 15, max number of iterations to 30, a tolerance residual of 1e-5, and tolerance iterations to 7 for every optimization. With these settings the PSO found the global minimum of the objective function for each coefficient pair using the decision variables as inputs.

The first two simulations focused on the two extremes of the weighting coefficients where ($C_{cost} = 1$, $C_{perf} = 0$) and ($C_{cost} = 0$, $C_{perf} = 1$). These represent two scenarios where minimizing the cost is the only objective and where maximizing performance is the only objective, respectively. Figure 5.50 shows the results of the optimal solution for minimizing cost without regard for performance, where the weighted coefficients are $C_{cost} = 1$ & $C_{perf} = 0$. This figure shows the low internet throughput transferring data from the server transfer queue to the cloud by the *Server Xfer Queue* in Figure 5.50(a). The low internet throughput is the cheapest package and if it takes data longer to get uploaded to the cloud the cloud storage costs will be lower. In Figure 5.50(b) shows the staging bucket data get deleted from cloud storage each time the data pipeline is run which also saves costs. This scenario finished the ETL process for all the data in 41 minutes and cost $0.09.

Figure 5.51 shows the second scenario where the weighted coefficients are $C_{cost} = 0$ & $C_{perf} = 1$. This scenario represents maximizing performance without regard to the cost. In contrast with the other scenario, this one finished in 3 minutes and cost $0.79. This is a 778% increase in costs, but a 93% increase in performance.

To aid in the sensitivity analysis later and to find a balance between these two extreme scenarios we ran 9 other scenarios through the PSO algorithm. Starting the first scenario with weighted coefficients of $C_{cost} = 0.1$ & $C_{perf} = 0.9$ we incremented C_{cost} up by 0.1 and C_{perf} down by 0.1 for the next scenarios until we go to $C_{cost} = 0.9$ & $C_{perf} = 0.1$. We did this as a method of MOO since the `pySwarms` PSO algorithm we used did not have the MOO capability.

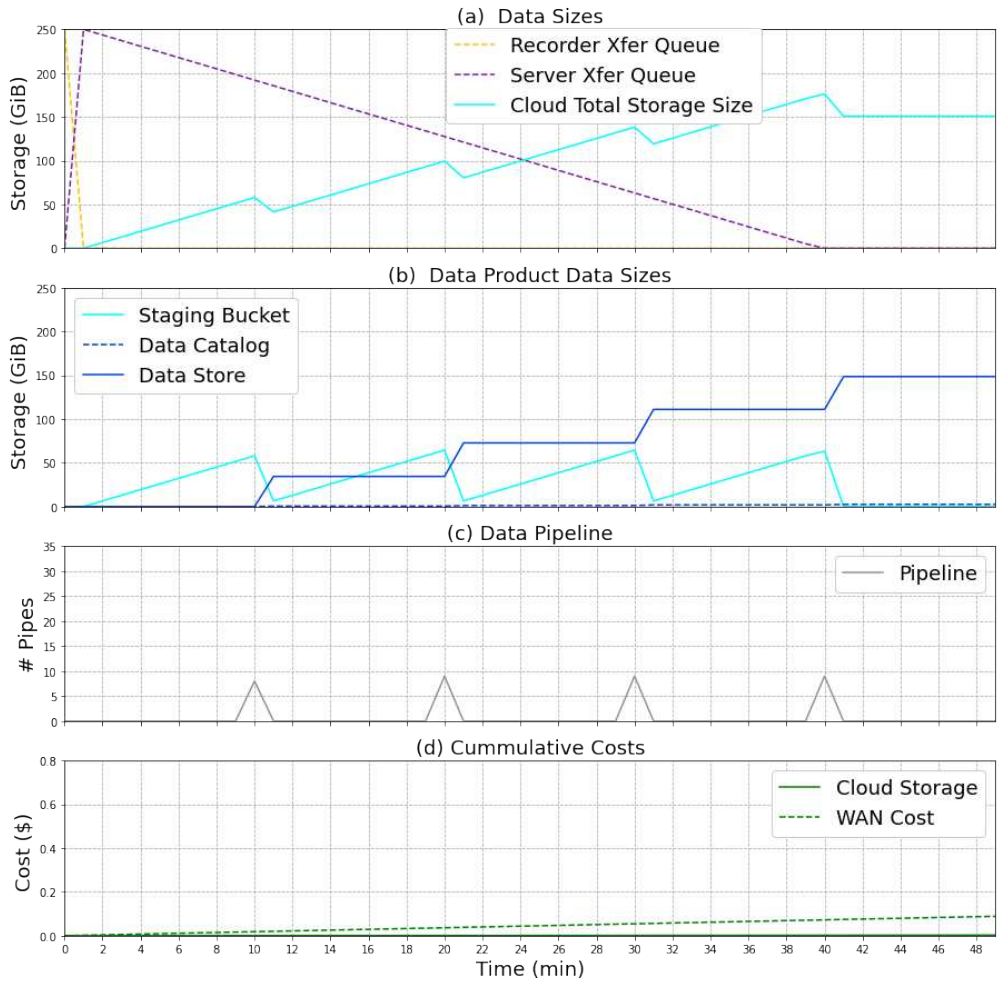


Figure 5.50: Time Series Plots of the Minimized Cost Data System (Model Ver.3)

For all 11 scenarios the PSO algorithm ran a total of 2,130 simulations with the Ver.3 UM which took 10 hours of CPU time to run. This averages 17 CPU seconds per run. The results of the 2,130 simulations are shown in Figure 5.52 and are plotted by the normalized cost and performance objectives. The Pareto-optimal solutions are highlighted by the Pareto front line.

The solutions in Figure 5.52 show at least three distinct trade space clusters, therefore we analyzed characteristics of the Pareto-optimal solutions and realized that the internet package was one of the dominant system characteristics. We plotted each of these internet packages a distinct shape and color to highlight it.

With a Pareto front, stakeholders can now choose a cost and know how to configure the system to get the optimal performance out of it. Optionally, they can choose the Pareto front vertex known

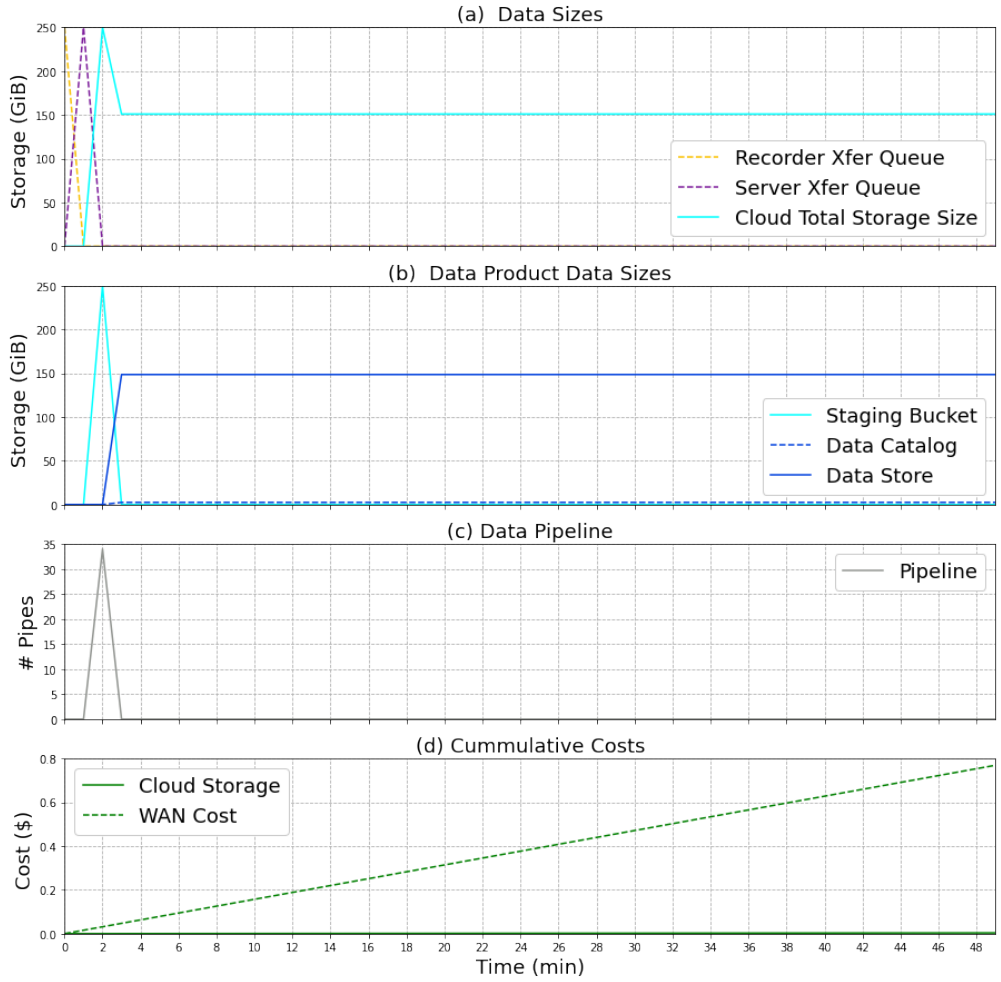


Figure 5.51: Time Series Plots of the Maximized Performance Data System (Model Ver.3)

colloquially as the "knee in the curve" and get the most balanced and optimal solution for their cost. Figure 5.52 shows this vertex at the far left and bottom of the solutions. At this point it would take more than double the cost to increase the performance slightly, while reducing the cost slightly decreases the performance drastically.

Figure 5.53 shows the time series data from Pareto-optimal solution at the Pareto front vertex. This data system used the mid-level internet package, deleted cloud data after processing, and ran the pipeline every minute there was data in the staging bucket process queue.

Table 5.8 shows the Pareto-optimal solution costs and performances of the two extreme scenarios and the Pareto front vertex scenario. This shows that the vertex solution only costs 3 times as

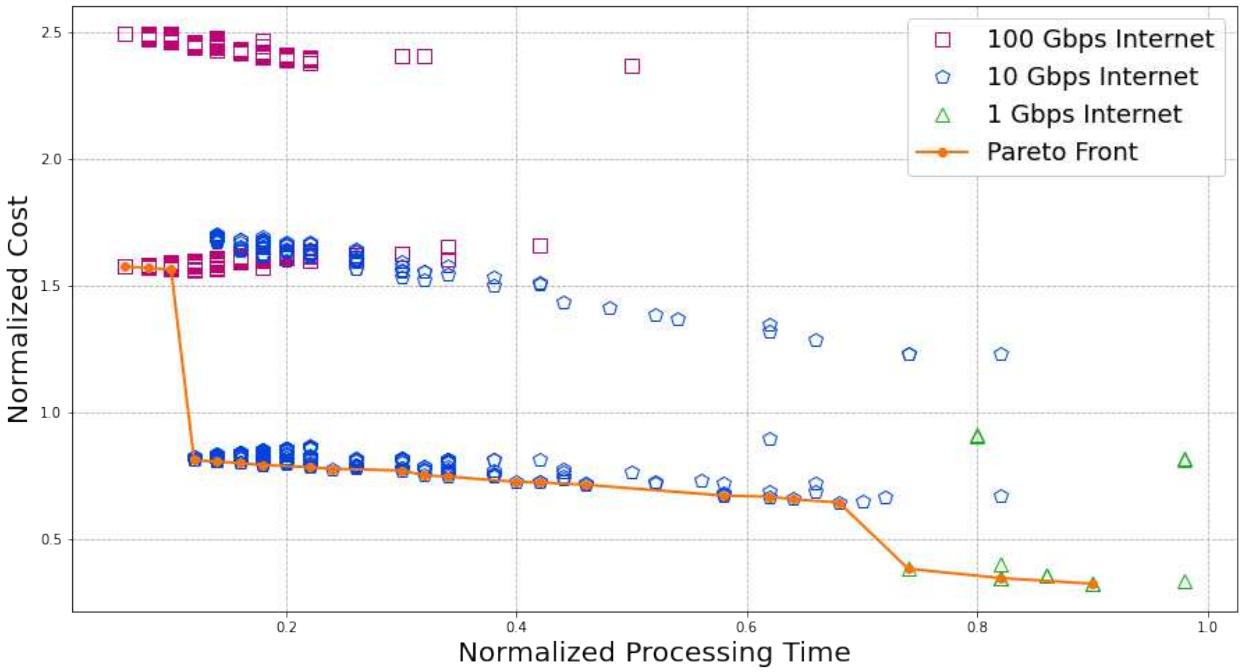


Figure 5.52: All Simulation Solutions with Pareto Front (Model Ver.3)

much as the minimal cost solution, but is almost 7 times more performant. Additionally, this vertex solution is a third of the cost of the maximized performance solution, but only half as performant.

Table 5.8: System Scenario Costs and Performance (Model Ver.3)

Scenario	Cost	Timing
Minimal Cost	$0.09	41 min.
Maximize Performance	$0.79	3 min.
Pareto Front Vertex	$0.27	6 min.

We discovered that the internet package is a dominant system characteristic for this simple Data Mesh data system. We also verified that our UMA is compatible with system optimization algorithms. This led us to the decision to continue to upgrade the UM model in complexity and fidelity of the system being developed.

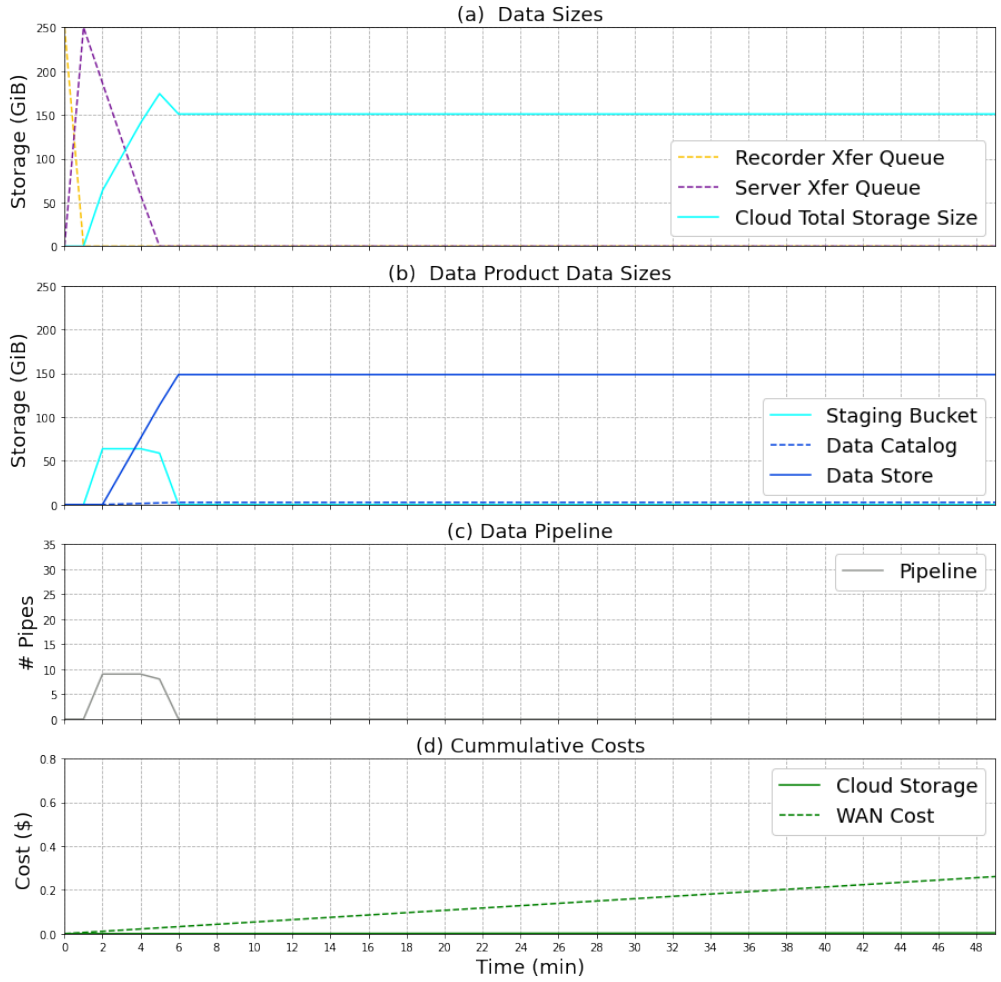


Figure 5.53: Time Series Plots of the Pareto Vertex Data System (Model Ver.3)

5.4.2 *Dev-V&V*

The major tasks that were accomplished in the *Dev-V&V* stage during the third YAMM iteration was the deployment of the data format 2 and 3 ETL pipelines to the operational data system and its Data Product. These align with MVPs 3 & 4 on the hybrid cloud schedule shown in Figure 5.5. In terms of the YAMM Process Flow 1 for these MVPs they are only half way through their first YAMM iteration. However, when considering the entire integrated data system we are halfway through the third YAMM iteration.

After the SW-Dev teams for MVPs 3 & 4 began deploying their ETL pipelines to the operational data system they began to provide feedback to the DI team. This feedback included the requests for more K8s resources defined within the SDI and other requests to fix defects in the DI.

This collaborative feedback led to the MVP 1 continuous DevSecOps development and deployment of the SDI by the DI team during this YAMM iteration.

The MVP 3 & 4 SW-Dev teams also had to integrate their ETL pipelines with the Data Product. The Data Product data warehouse and datastore paradigm was designed and developed by MVP 2 SW-Dev team. Collaborative and three-way feedback by the MVP 2, 3, & 4 SW-Dev teams led to the continuous DevSecOps development and deployment of these MVPs during this YAMM iteration.

5.4.3 *Operational Data System*

As we similarly discussed in Section 5.4.2 the majority of the activities in this third YAMM iteration for the *Operational Data System* consisted of continuous feedback, collaboration, and DevSecOps software updates and deployment to the operational data system environment. End user feedback continued to drive some of these software updates as well.

Continuous AWS cloud resource data and system metadata collection also happened throughout this iteration. However, after initial data analysis on the resource data we found out that the rates of data collection and the number of days this data was saved was not sufficient. Some data was being recorded at one minute intervals, but would only be saved for 14 days. After this it was aggregated and saved for daily intervals. The lack of data at an hourly rate saved for at least 6 months reduced the fidelity of the analytical models we could generate. This was not a serious issue since we still have many months of data to gather after this.

The solution to this issue was to enable the AWS paid service to record and save this data at the rate we requested. This was accomplished near the beginning of this third YAMM iteration and this is the driving factor behind using this time frame for data analysis and model verification.

5.4.4 *Data Analytics & Algorithmics*

In this section, we discuss an overview of the data analytics performed on the resource data and system metadata to identify latent variables, resource constants, algorithm variables, and the data to perform V&V on the UM. The V&V data that we extract fit into three categories, which

are time-series input data, initial system conditions input data, and the time series validation data to measure the UM's accuracy.

For empirical data feedback into the next UM upgrade and V&V, we chose to limit our analytics to S3 storage size and cost. S3 costs account for over a third of the cloud costs of our prototype system and which the current Ver.3 model is relatively close to simulating. Figure 5.54 shows the percent costs of each of the six highest cloud service costs during the time frame we analyzed. This figure illustrates that S3 costs would make the biggest impact to cloud cost if optimized.

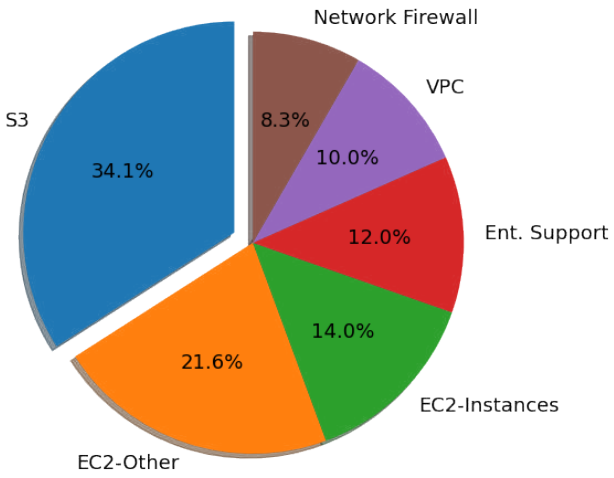


Figure 5.54: Data Mesh System Prototype Service Cost Percentages

The cohort analysis and simulation constants developed during this stage were based on the Ver.3 UM discussed in Section 5.4.1. Specifically, we only extracted resource data to quantify the storage behavior and cloud S3 storage cost behavior for more accurate simulations and system optimizations presented in Section 5.4.1.4.

During our analysis, the latent variable most impactful to our accuracy was the storage cost rate. This rate is set by the CSP, however we found that our previous estimations were inaccurate when using the standard storage cost rate from the AWS website. The linear relationship between storage size in GiB and storage cost in our empirical data showed a different storage cost rate than advertised. This difference caused large errors in our simulations which would have caused large

errors in future cost estimations. We were not allowed to know if we received a different cost rate than the posted rate on their website, therefore we used this empirically derived storage cost rate to account for these unknowns.

The resource constants we extracted from the data include data pipeline compression ratio, single pipeline throughput, data warehouse proportion, datastore proportion. The algorithm variables we derived from the system and empirical data include pipeline schedule queuing behavior, raw upstream data deletion behavior, and parallel processing behavior.

Before we performed another system optimization with new data, we decided that the UM needed to undergo V&V. To perform this V&V we had to create a simulation-enabled model with the capabilities of the Ver.3 and the physical system specifications from the Ver.1 model. Therefore, we created a Ver.4 model which is the first simulation-enabled model which includes the simulation libraries and the hybrid cloud prototype specifications. Figure 5.14 shows the targeted logical & physical abstraction of the prototype specifications that the Ver.4 is built around. This model was developed in parallel with the data analytics in this stage. We will discuss the details of the model in the next YAMM iteration section. The operational system V&V resource data extracted in this stage was targeted for this Ver.4 model.

5.4.4.1 V&V Time Series Input Data

This section provides an overview into the V&V model time series input data and the methods we used extract it from the AWS resource data. One of the goals of the UMA is to enable a model to accept time series arrays as input. This is one way to estimate software performance on different hardware at the scale we are simulating [19]. It also provides a means to run different scenarios that vary in time, rather than the static bulk upload of data we used in the system optimization scenarios in Section 5.4.1.4. Time series data input can also capture manual behavior not easily modeled or behavior that is not Gaussian. The time series input also enables a practical method of performing V&V on the UM.

The data we collected and processed for the Ver.4 model validation included the initial conditions for the start date of the simulation, time series input, and time series empirical data to

measure the model output against. The initial conditions include constants for each UM structural element's value properties such as the the local server, Snowball, and Daqscribe upstream data sizes and transfer queue sizes, the staging bucket upstream size, processing queue size, a cloud storage cost, and finally Data Product processed data sizes in the data catalog and datastore. For this V&V effort, we assumed that the data from the recorder had already be fully transferred to either the local server, Daqscribe, or Snowballs. We did not collect on-prem resource data, so we were unable to measure the throughput or timing of data transfer from the recorder.

The time series inputs that we collected include the data throughput (Gbps) from each of the Daqscribe, local server, and Snowballs, the throughput from the staging area through the Data Product. These time series throughput inputs are used to mimic the very specific real-life scenario that we are validating the model output against. This scenario included manual starting and stopping of the data pipelines, continuous ETL software updates and deployments, and some downtime of the DI as it was also upgraded. This type of logic is not built into the model, rather we use these time series inputs to approximate this scenario.

The time series validation data that we collected and processed includes the storage sizes, queues, and cloud storage costs of each SysML structural element in the system and also the rollups of these values to the system level.

To collect the model time series throughput from the on-prem hardware to the cloud we had to extract this data from the cloud-based staging bucket S3 metrics. We calculated daily throughput by the change in daily storage size of the staging bucket. This throughput input for the three data paths to the staging bucket shown in Figure 5.14. Since these S3 metrics did not have system metadata to tell us where the data came from, we had to estimate throughput rate upload to the cloud for the three on-prem data sources. We did this by first normalizing the S3 metrics data, and then applying a Bayesian Gaussian Mixture clustering algorithm to assign the different throughput cluster rates to the three upstream data sources. Figure 5.55 shows this normalized throughput data tagged by the most likely data source and the clustering algorithm's confidence ellipses.

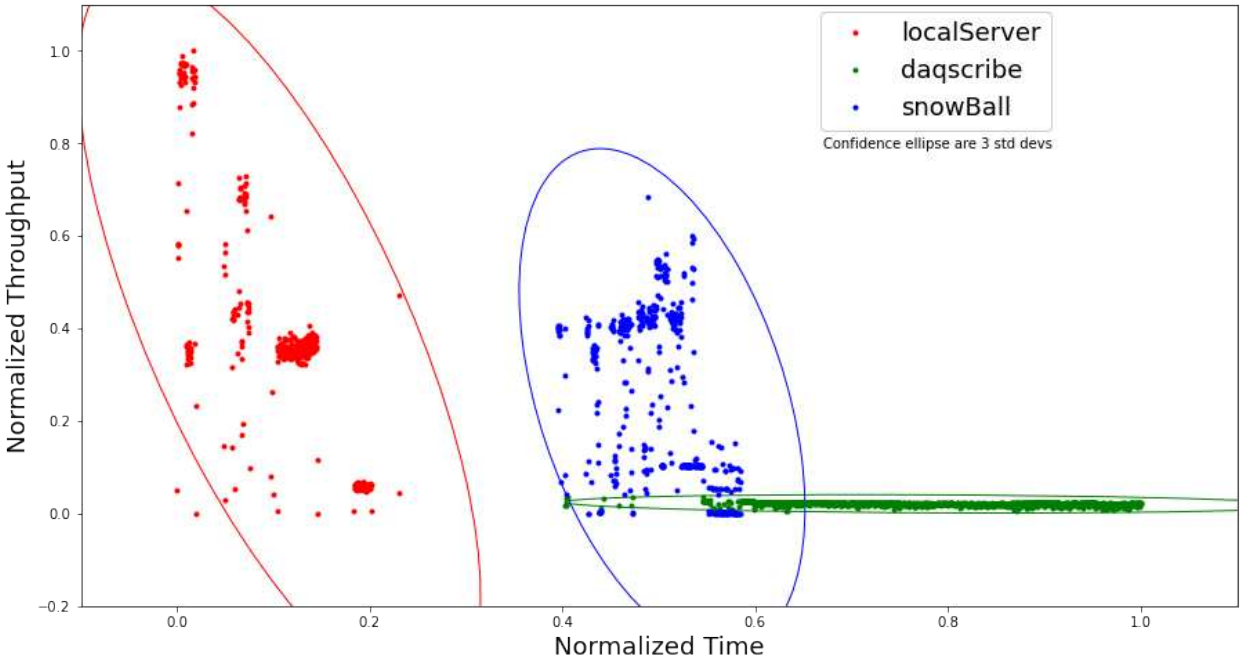


Figure 5.55: On-Prem Device Throughput Clusters (Normalized)

Figure 5.56 shows the simulation input provided to the UM. These time-series on-prem to cloud throughputs are used as the maximum throughput value property (*maxThroughputOut*) in the model. The large fluctuations in throughput for the Local Server and Snowball in this figure are a result of manual data transfers happening as the end users had time to upload data. The Daqscribe data upload was automated and is shown to have a steady, albeit slow, throughput.

Figure 5.57 shows the staging bucket to Data Product throughput data used as model input for the V&V scenario simulation. Comparing this figure to Figure 5.56 shows that the data is processed by the ETL data pipeline happens after the majority of the data has been aggregated in the staging bucket half way through month 12. This throughput represents all three data formats and the UM simulates them as one single ETL pipeline for this V&V effort to simplify this initial model. Additionally, the throughput in this figure only represents the final ETL software versions of each data format as they were deployed to the operational data system environment. Previous software version data processing was not considered for this initial V&V effort in order to simplify the process. We discuss how we accounted for this in the datastore data sizes and costs below. The

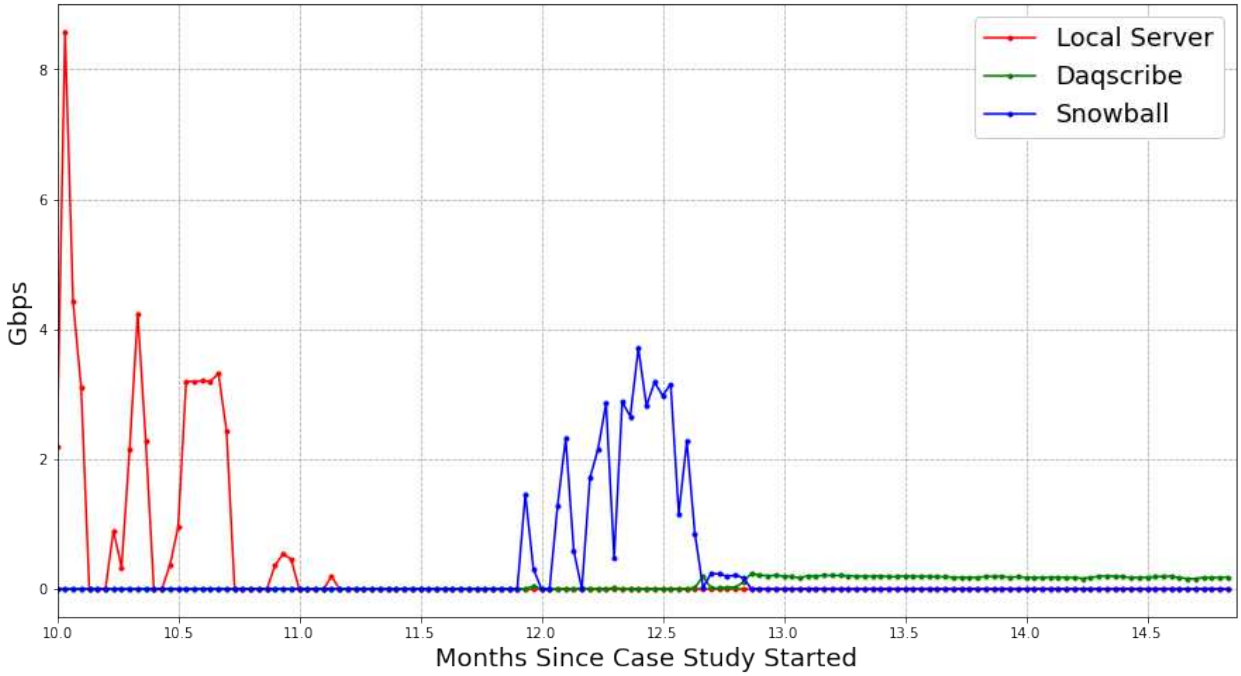


Figure 5.56: On-Prem Devices to Cloud Throughput Input

large fluctuation in throughput is due to many variables such as each final ETL script starting at different times, new data being uploaded to the staging bucket, and some resource limits being hit.

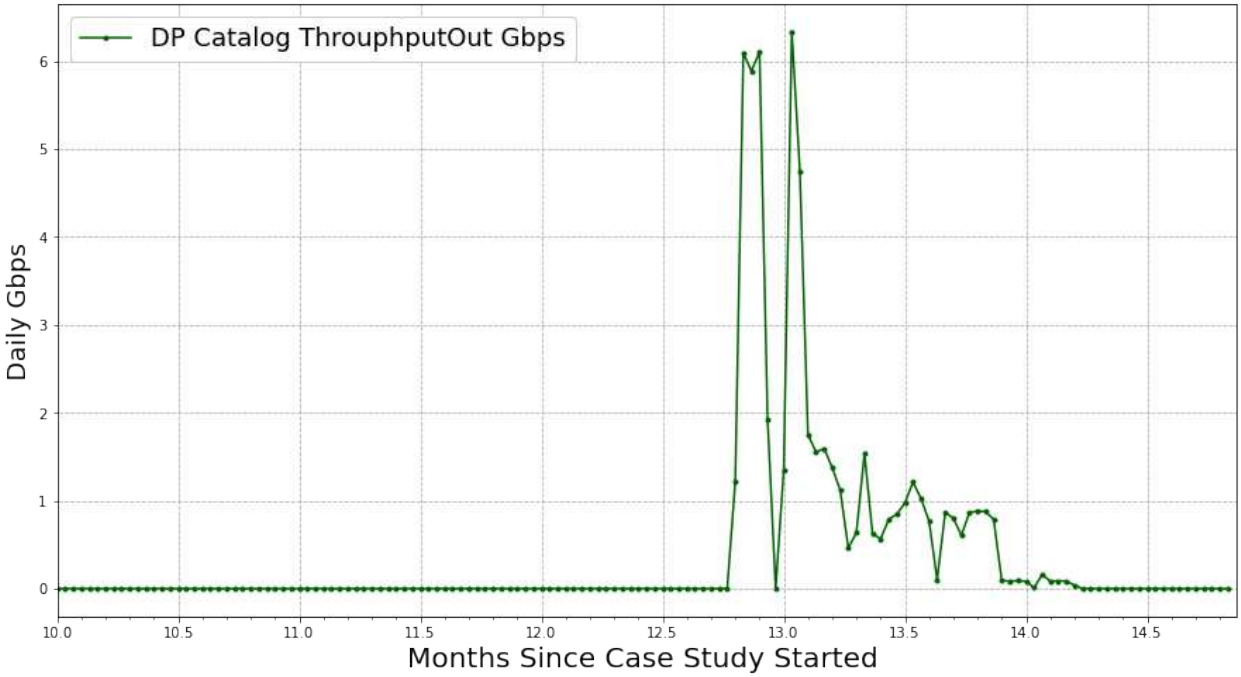


Figure 5.57: Staging Bucket to Data Product Throughput Input

5.4.4.2 Time Series Validation Data

This section provides an overview into the V&V model time series validation data used to measure the model's accuracy and the method we used extract it from the AWS resource data.

We had two tasks of data processing to format the AWS cost and resource data to enable the V&V of our model. The first task involved separating the cloud storage cost of this case study from another team's work happening in a different area in the same AWS environment. The second data processing task involved removing the extra storage data and costs associated with the continuous software development and deployment of 15 different ETL software versions across all 3 data formats. We performed these two tasks to simplify the modeling effort for this YAMM iteration.

The first data processing task was related to AWS's method of charging S3 storage costs. We needed the S3 costs as a function of time for a single S3 bucket, but AWS only provides cost data for the entire account and there were three S3 buckets within our account. The other 2 S3 buckets were not related to this case study, therefore we extracted the proportions of these extra S3 bucket costs from the total account cost to estimate our staging bucket cost to validate our Ver.4 UM. Figure 5.58 shows the Data Account total S3 size and the three S3 bucket sizes within that account. One of these buckets uses glacier storage half way through, but this is still considered the same bucket and the same amount of data. Figure 5.59 shows the staging bucket's S3 cost proportion that was used for model validation.

The second V&V data processing task was applied to the Data Product Account. The Data Product Account only contained a single S3 bucket, so it did not need a proportional cost adjustment. However, this bucket was used as MVPs 2, 3, & 4 software deployment target and received 15 different continuous software deployments. Unfortunately, this meant that a new data catalog and datastore were created for each software version resulting in 15 different versions datastores. This is not a scenario we wanted to model initially, so we corrected for these different datastore version by removing them and their costs from the Data Product account and bucket.

The datastore sizes for each data format and ETL software version are shown in Figures 5.60, 5.61, and 5.62. There was one final software version released for data format 2, but we do not

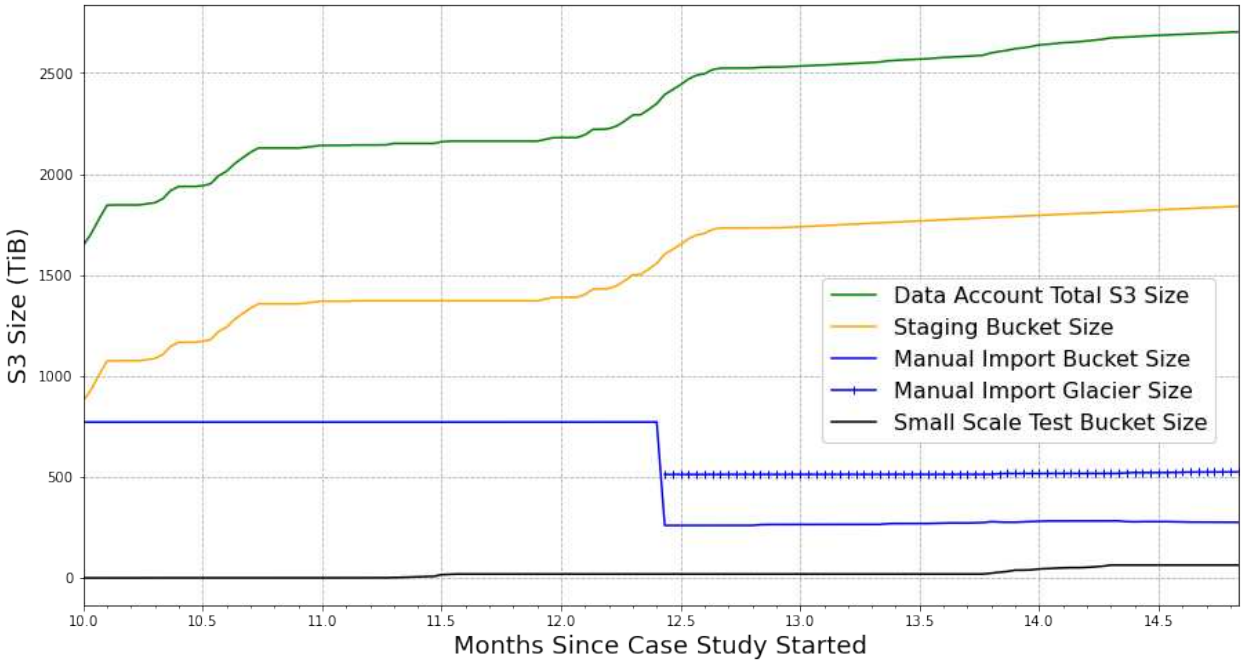


Figure 5.58: Data Account and S3 Bucket Sizes

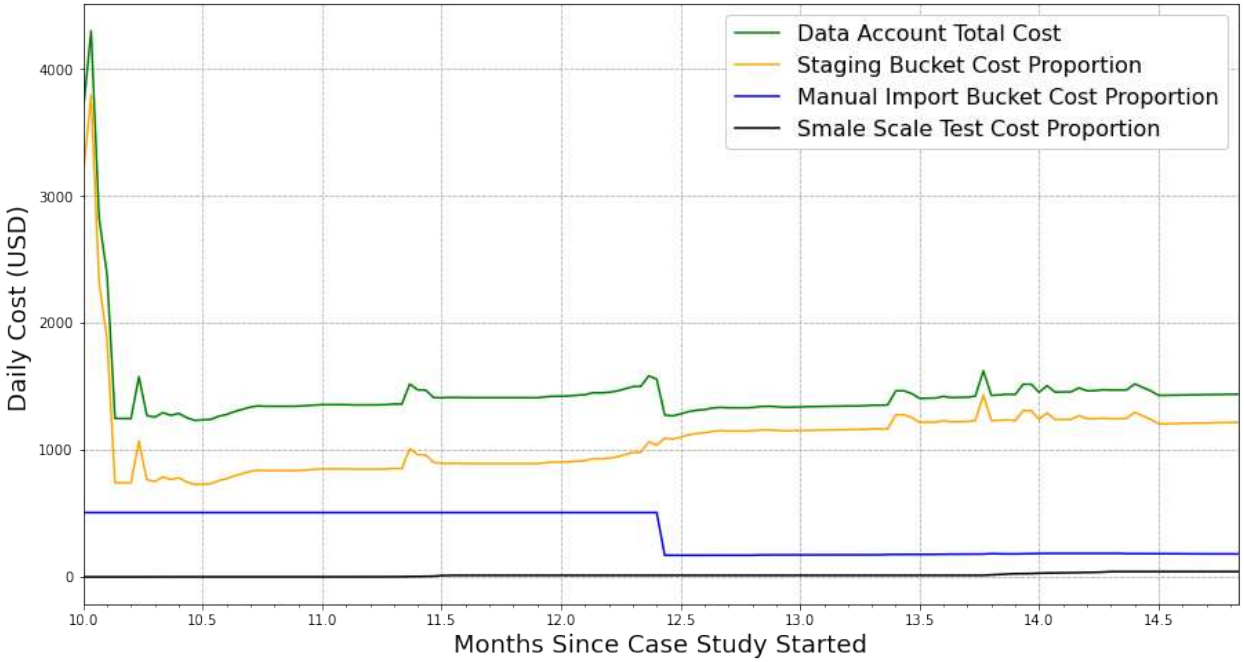


Figure 5.59: Data Account and S3 Bucket Cost Proportions

show this in Figure 5.61 since it was released outside of the case study data analysis window for this V&V task. All Figure 5.63 shows the original Data Product S3 bucket size and the bucket size after adjusting for the incorrect datastores.

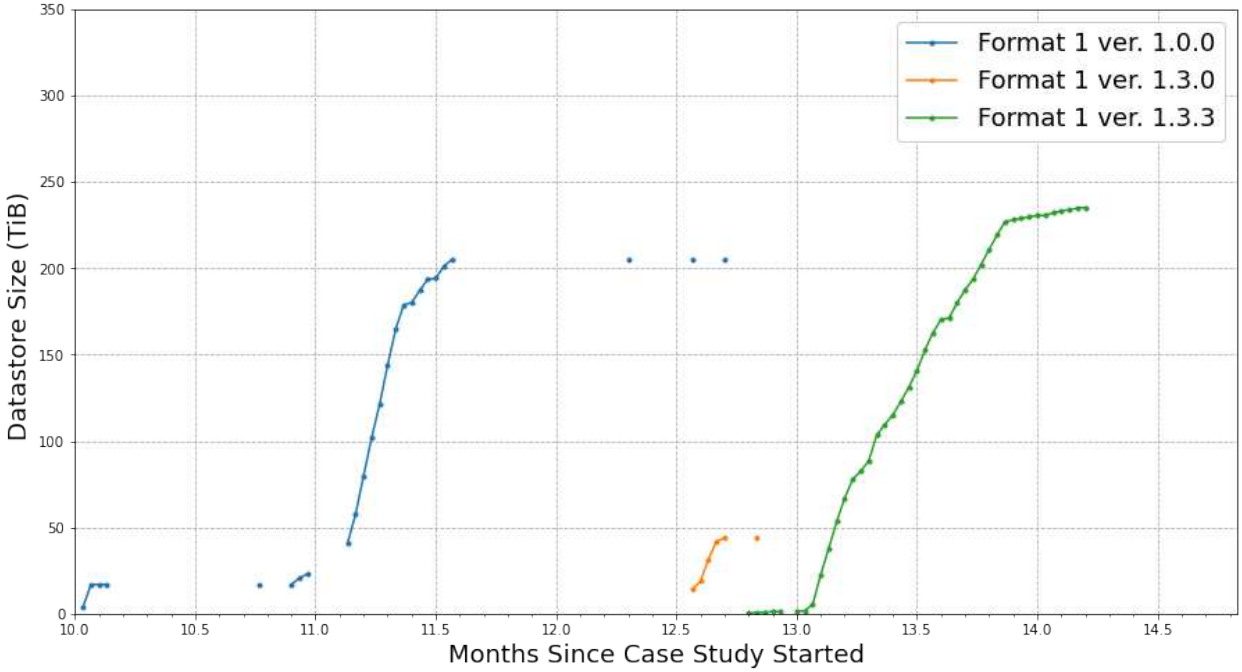


Figure 5.60: Data Format 1 Datastore Size by Software Version

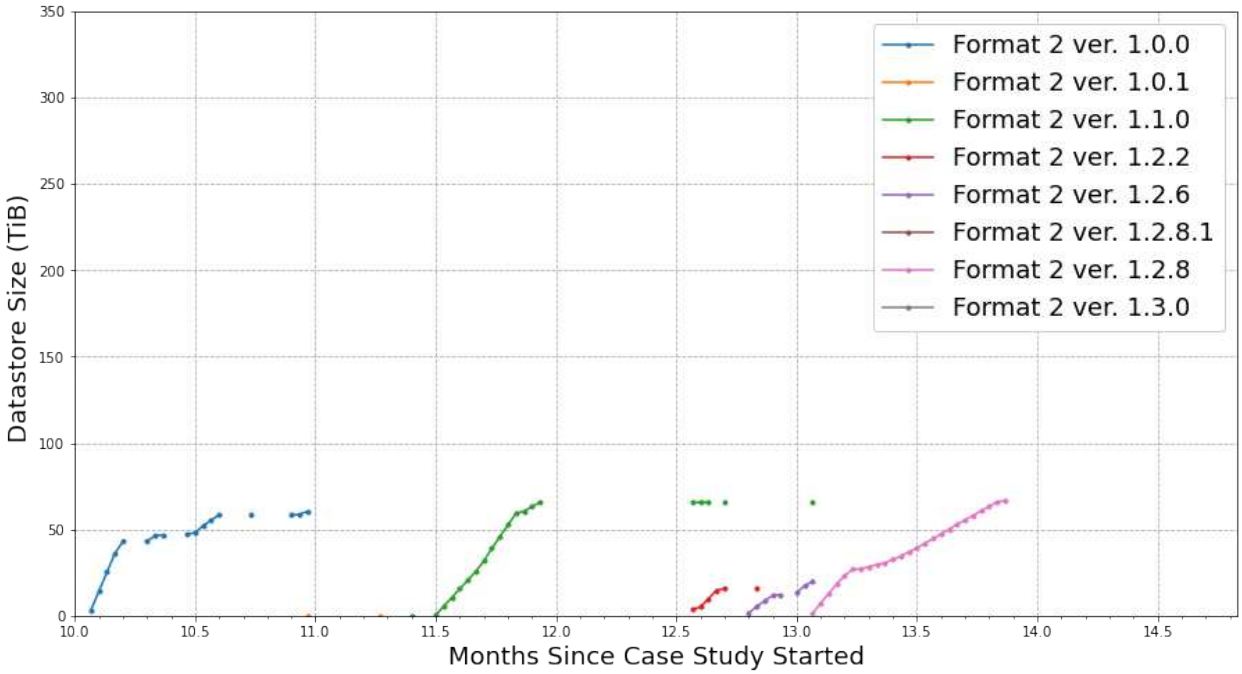


Figure 5.61: Data Format 2 Datastore Size by Software Version

In addition to the incorrect datastores, we also had to correct for a significant amount of data being manually deleted from the Data Product S3 bucket on two occasions. This was old data that

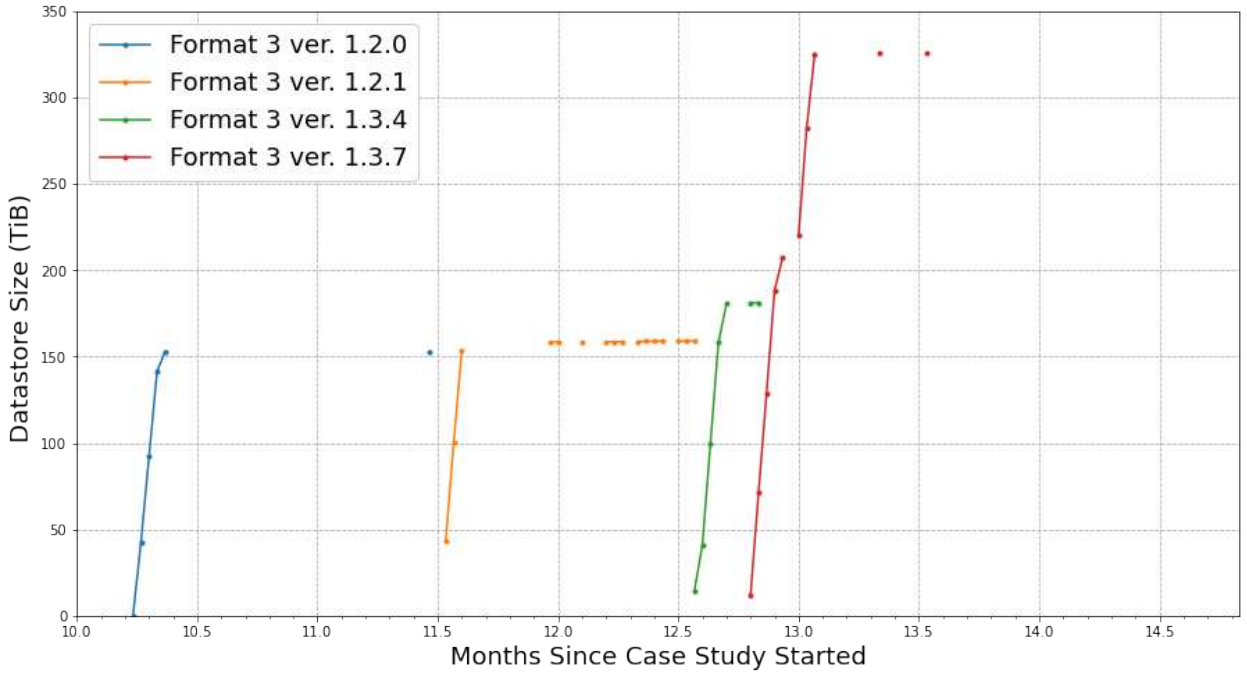


Figure 5.62: Data Format 3 Datastore Size by Software Version

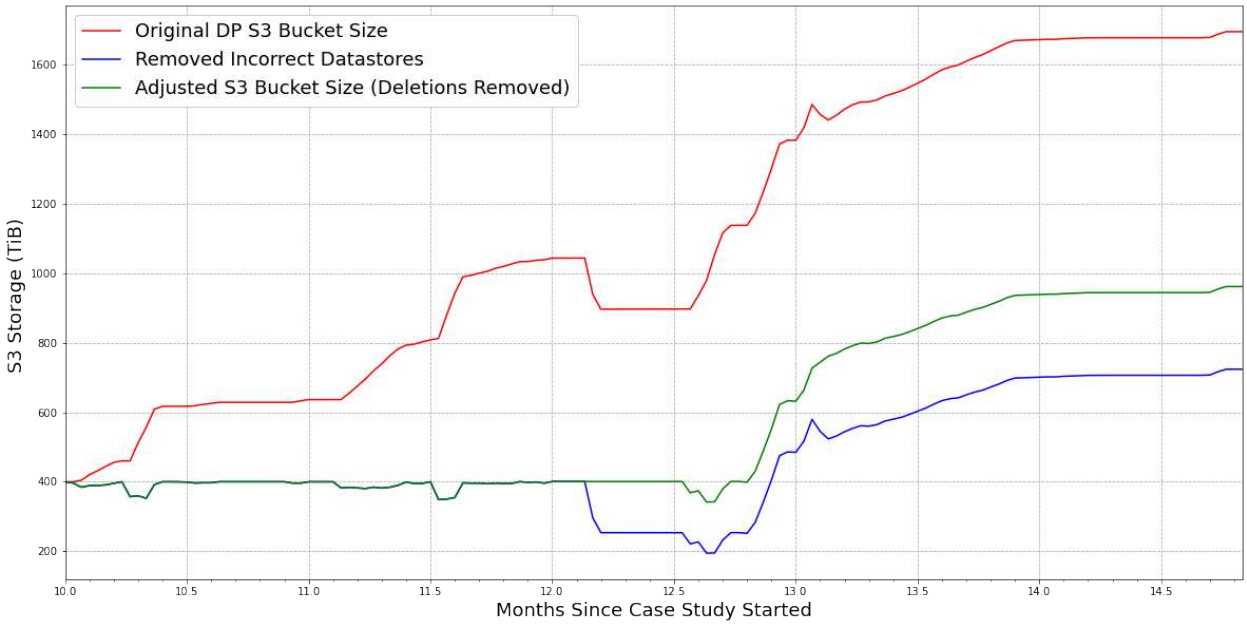


Figure 5.63: Data Product S3 Bucket Size and Corrections

was present before the time window of this analysis. We adjusted the size and costs as if this data was not deleted manually because this is not a scenario or set of logic we wanted to build into the UM model at this time.

Figure 5.63 shows the final bucket size after adjusting for the manually deleted old data. Figure 5.64 shows the original Data Product S3 bucket cost and the final bucket cost after accounting for the two data size adjustments.

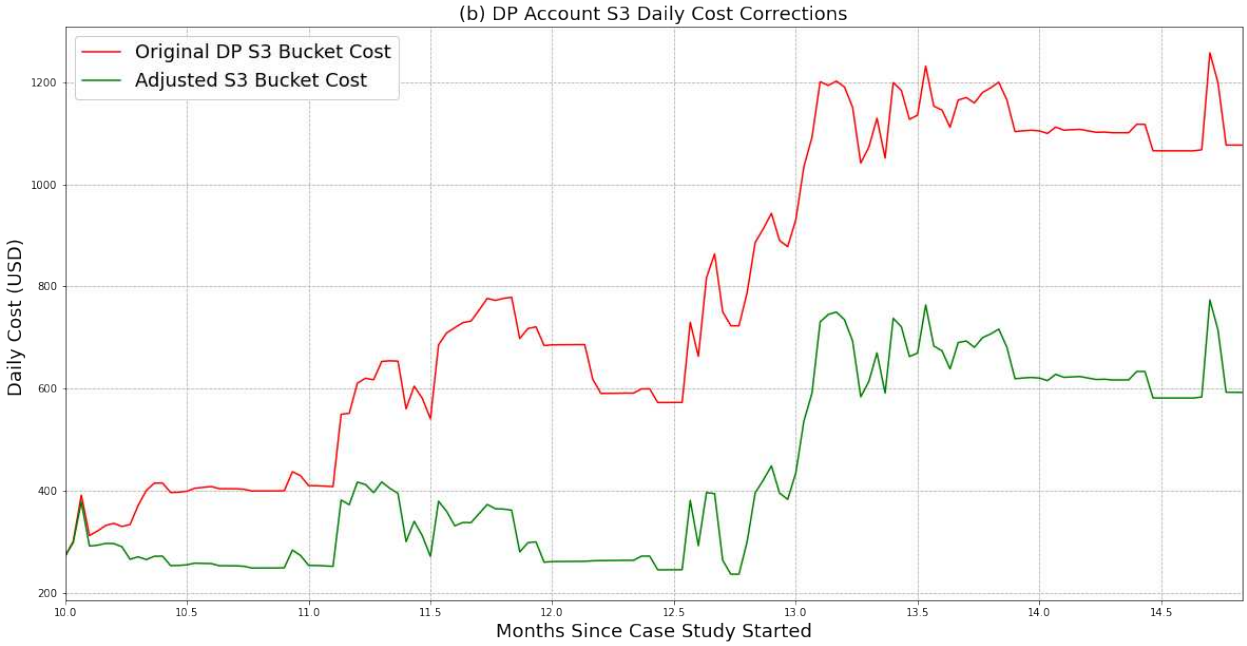


Figure 5.64: Data Product S3 Bucket Cost Adjustments

5.5 YAMM Iteration 4

This section describes the final full YAMM iteration which was comprised mostly of the upgrade and V&V of the Ver.4 UM, the continuous collection of resource data, and the analysis of resource data to provide empirical feedback for the Ver.5 UM update, V&V, and system optimization. In Section 5.5.1, we provide an overview of the Ver.4 UM that includes the hybrid cloud prototype AWS physical specifications and logical Data Product specifications and implements the upgraded simulation libraries for predicting storage sizes and costs. This section also includes the V&V of this upgraded model using empirical data collected and processed during the last YAMM iteration.

In Section 5.5.2, we wrap up the last of the MVPs being developed and deployed. In Section 5.5.3, we summarize the work to analyze and extract component information for the cloud compute, cloud networking, recurring cloud service costs, and purchases for the entire hybrid cloud data system. We also extract the V&V data to ensure the upgrades to build the Ver.5 model are validated.

5.5.1 *MDAO*

In this section we describe the upgrades made the Ver.4 UM by merging the Ver.1 specification model and the Ver.3 simulation-enabled model. We discuss these upgrades focused on increasing the speed of the model and reducing the Random-Access Memory (RAM) required for each simulation in Section 5.5.1.1. In Section 5.5.1.2 we present the results of the V&V of this new UM using the data produced from the last YAMM iteration.

5.5.1.1 UMA Ver.4 Model Upgrade

As shown in Figure 5.4, we merged the initial architecture and design model (Ver.1) and the simulation-enabled simple Data Mesh model (Ver.3) to create the Ver.4 simulation-enabled Data Mesh storage UM. Figure 5.3 shows how we reused the modular library models for this new model. This figure also shows that we reduced the direct dependencies of the library models from five direct dependencies of the Ver.3 model to two. We created the Data Mesh Library to provide a single model library framework that manages all other data system UMA libraries. This reduced the workload on the modeling team when starting a new project and also improves the organization of the model libraries.

UMA Metamodel Visual Specification Layer

The first upgrade to the UMA model framework we accomplished was to split the previous Abstract Sub-System block, shown in Figure 5.29, into separate abstract objects as shown in Figure 5.65. We broke the Sub-System block into six abstract objecta, which allows us to reduce the number of inherited properties for each element in the model. Fewer properties in each element

increased simulation performance and reduced RAM used by the UM during the TWC server-side execution.

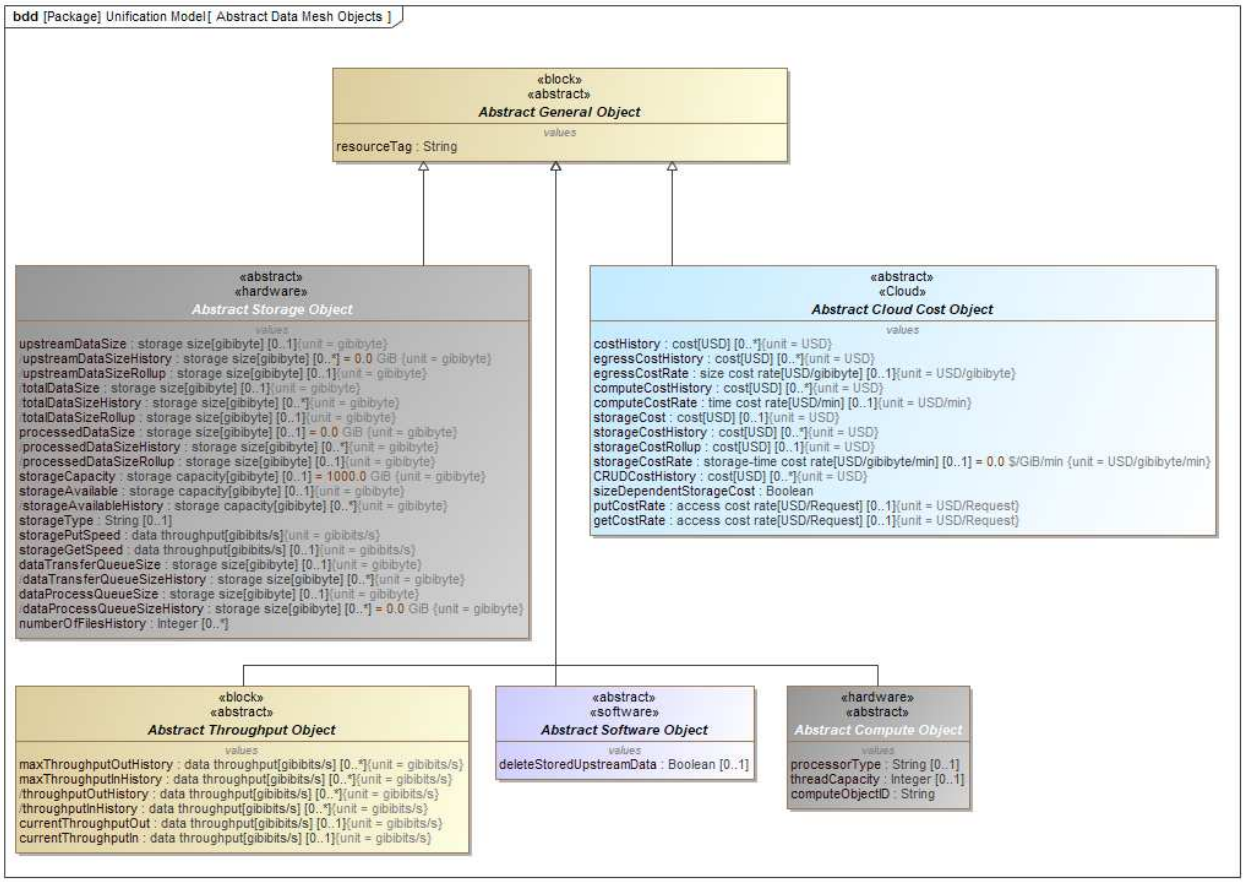


Figure 5.65: Abstract Data Mesh Objects (Model Ver.4)

The visual specifications we kept from the Ver.1 model are shown in Figures 5.14, 5.15, and 5.16. Each of the four use cases in Figure 5.16 was developed to include simulation behaviors and opaque behaviors to enable simulation. The upgraded Data Product behavior and associated simulation from the Ver.3 model shown in Figure 5.35 was also added to this Ver.4 model.

We also added visual specifications to automate the transfer of data from the recorder to the local server or Snowball. Figure 5.66 shows additional specifications we added to ensure software scripts are written to automatically download data from the recorder to the local server. A similar specification was created for the Snowball.

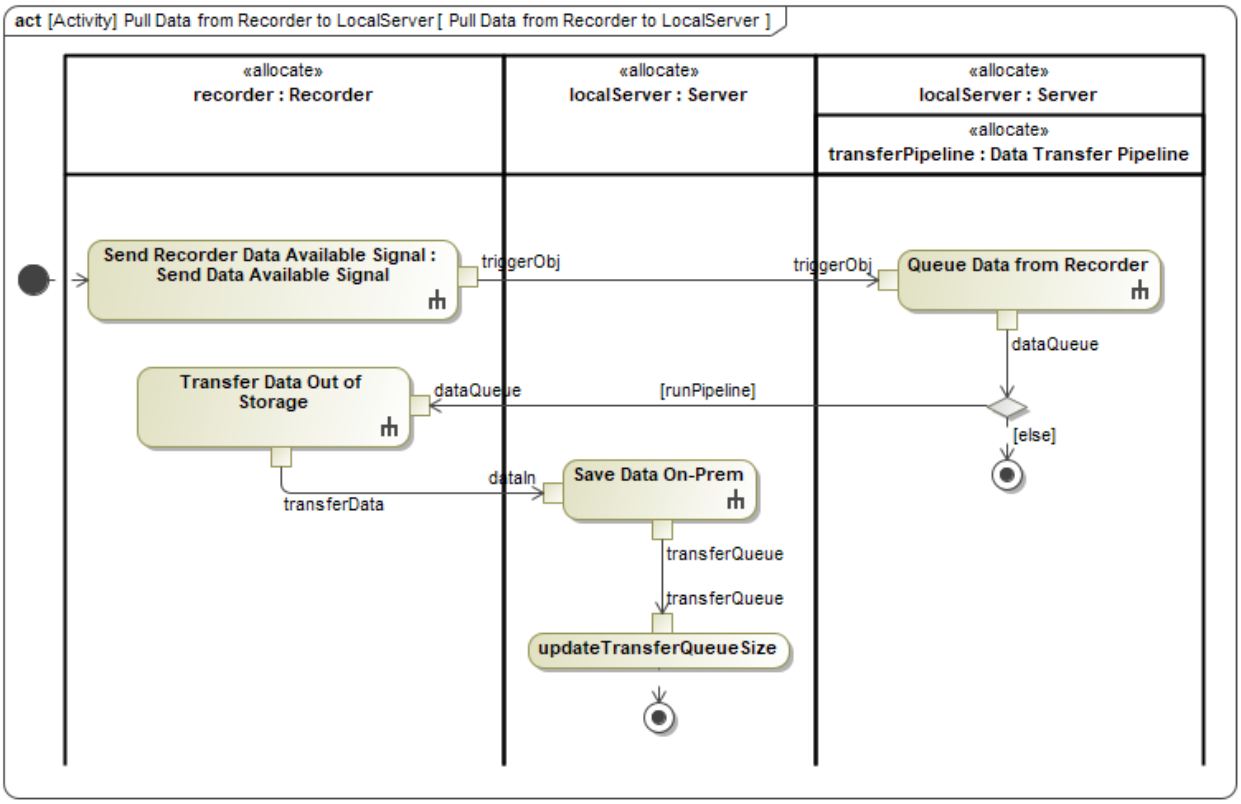


Figure 5.66: Automated Download of Data From Recorder (Model Ver.4)

UMA Metamodel Visual Simulation Layer

Our new simulation block for the primary use case in this model inherited the abstract simulation definition. We also created a simulation behavior similar to Ver.3 to create the Ver.4 simulation definition and behavior shown in Figure 5.67. This simulation behavior uses the primary use case definition shown in Figure 5.16.

UMA Metamodel Functions & Utilities Layer

We used the Data Mesh model library framework for the Ver.4 model which then inherited the full set of libraries shown in Figure 5.49. This enabled the Global Groovy Class and associated opaque behaviors with these libraries to be easily reused in the next model. During the V&V phase defects were found, which led to updates to many areas of the Data Mesh model library framework resulting in better quality libraries.

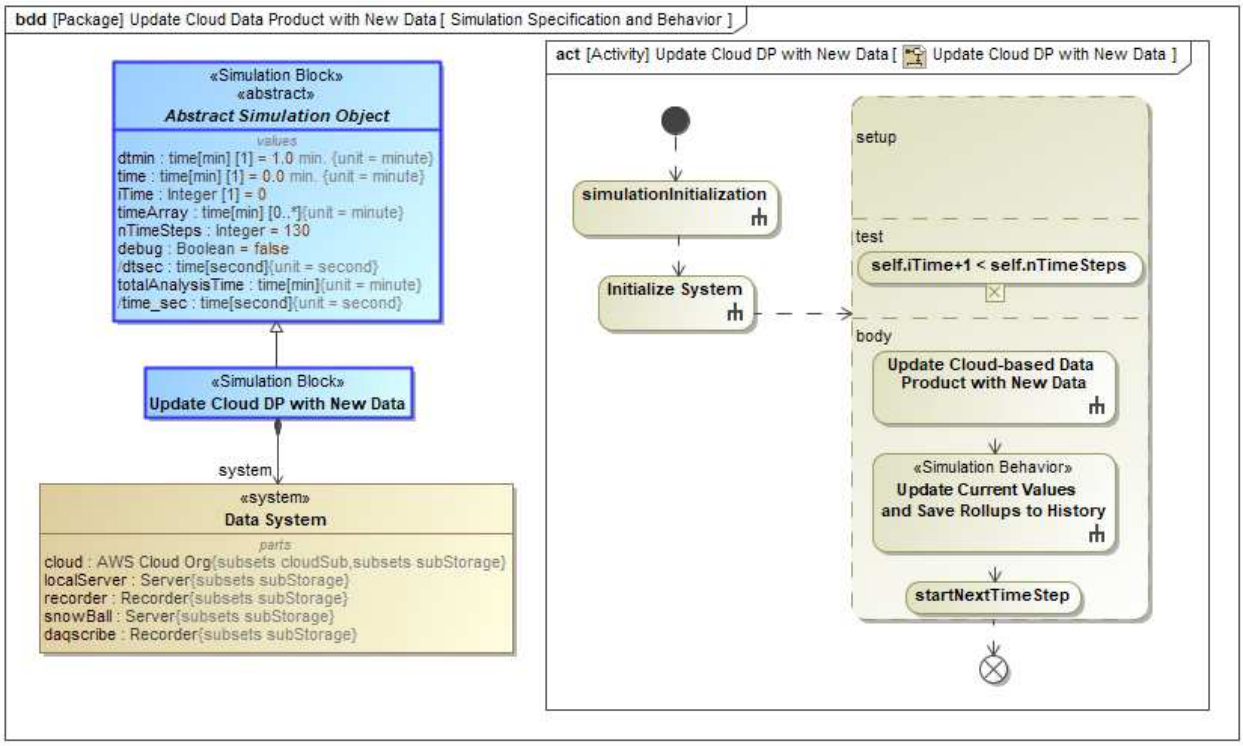


Figure 5.67: Simulation Definition and Behavior (Model Ver.4)

5.5.1.2 Model Ver.4 V&V

This section summarizes the UMA V&V of the Ver.4 model using empirical data as the input and validation for the simulation output. The results of this V&V effort are shown as time series plots and quantitative measurements as discussed in the UMA in Section 3.1.1.5.

As defined by our UMA, we wrote unit and functional tests for the time-series simulation output of the Ver.4 UM to ensure our combination of specifications and simulations were functioning correctly. Examples of our verification tests include: 1) Verify each time-series array is initialized and never exceeds the number of simulation time steps, 2) Verify data throughput history values do not exceed the maximum throughput, and 3) Verify each storage available array value is equal to the storage capacity minus the total data size of each storage object. These unit and functional regression tests were run each model updated.

We designed UMA to provide the novel capability of empirically-based quantitative validation of the UMs. Table 5.9 contains the quantitative accuracy measurements of the UM simulation

results compared to the data system prototype empirical data. The table shows the AWS accuracy of the cost and storage size of the S3 staging bucket, the Data Product, and the total cloud. Each of these data system components are shown Figure 5.14. This table also includes the total S3 cloud size and cost for a direct comparison with the simulation values.

Table 5.9: Quantitative Measurements of Accuracy (Model Ver.4)

Variable	Percent Error	RMSE	Residual Entropy	Max Entropy Percent Error	AWS Total	Sim Total
Staging Bucket						
S3 Size:	1.6%	5.5 TiB	1.9	60.0%	1,839 TiB	1,869 TiB
S3 Cost:	4.8%	$416.17	1.2	73.0%	$158,856.56	$151,300.74
Data Product						
S3 Size:	4.0%	10.1 TiB	2.3	51.0%	961 TiB	1,000 TiB
S3 Cost:	8.0%	$83.88	3.6	23.0%	$63,194.66	$58,145.97
Total Cloud						
S3 Size:	2.5%	11.8 TiB	2.8	40.0%	2,800 TiB	2,869 TiB
S3 Cost:	5.7%	$444.36	1.8	61.0%	$222,051.22	$209,446.72
Max Entropy: 4.6						

The state of the Cloud Report [12] listed cloud costs that were 18% over budget for the year. This helps to provide context to the cloud cost percent error values in table Table 5.9. In this context, our total percent error of 5.7% for our first UM V&V is acceptable.

We use RMSE as a quantitative measurement instead of MSE so that the answers can be represented in the same units as the values being measured. This provides more context to the reader than MSE. For comparison, the data size RMSEs were 2 OOMs smaller than the total data size. The cloud cost RMSEs were 3 OOMs smaller than total cost.

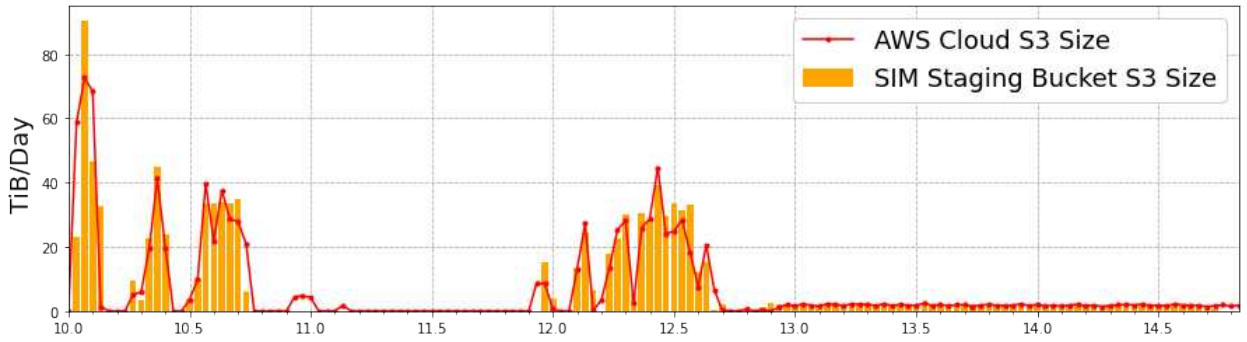
In Table 5.9 we calculated residual entropy to quantify the amount of non-random residuals in the simulation results. Larger numbers of residual entropy imply higher model accuracy. In other words, according to Simske [72], "when all we have left between real and simulation is randomness, then we have simulated the actual information". We used (3.3) to calculate the residual entropy listed in the table. We present the entropy, max entropy, and percent max to provide a

quantitative measure of the non-random behaviors the simulation failed to capture. In this case our total cloud storage and cost max entropy percent errors of 40% and 61%, respectively, indicate there are non-random storage and cost behaviors not captured in our simulation. These results indicate there could be additional work to improve this simulation's accuracy, however additional analysis is needed to ascertain the reasons behind these errors.

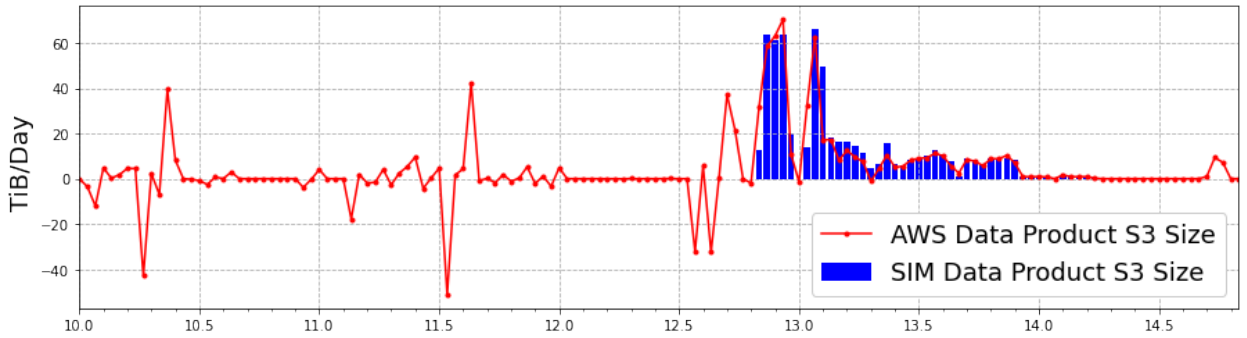
Figure 5.68 shows the time-series cloud storage size comparisons of the simulation and the real data system prototype. Figures 5.68a and 5.68b show the daily cloud storage size changes for the staging bucket and Data Product S3 sizes, respectively. Figure 5.68b shows large spikes of data fluctuations, both positive and negative, throughout this time frame. With the presence of large negative data size spikes, we attribute the majority of these fluctuations to manual SW-Dev CRUD actions. The low simulation residual entropy results are most likely attributed to these human induced fluctuations. This model version was not built to predict these manual actions, but could be upgraded to do so. Figure 5.68c shows the cumulative cloud storage for these two S3 storage bukets.

Figures 5.69a and 5.69b show the daily cost comparisons between the two S3 buckets. Figure 5.69c shows the total cumulative cloud cost comparisons against the two S3 buckets.

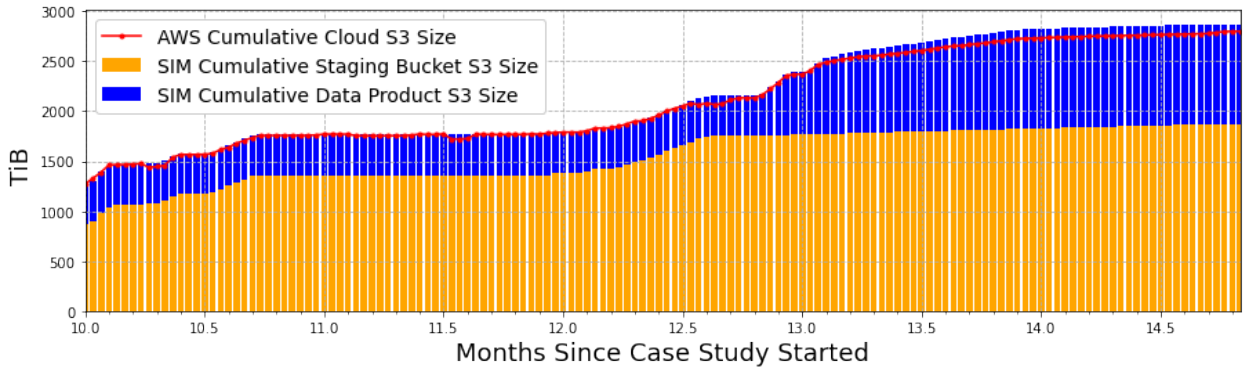
The results show that our Ver.4 UM accurately predicted the cloud storage and cost compared to a real system. Additionally, the source of the majority of fluctuation errors was found and was specifically not modeled for this initial effort. These human induced fluctuations could be accounted for in future models through Gaussian estimates. In addition to the model's accuracy, we demonstrated that our UMA could create a single UM with the five key modeling capabilities lacking in other MBSE approaches. The accuracy and ability to capture emergent system behavior indicate that our UMA and this model is capable of accurate system optimizations of complex software-intensive data systems.



(a) AWS vs Sim Daily Staging Bucket S3 Size (Model Ver.4)



(b) AWS vs Sim Daily Data Product S3 Size (Model Ver.4)

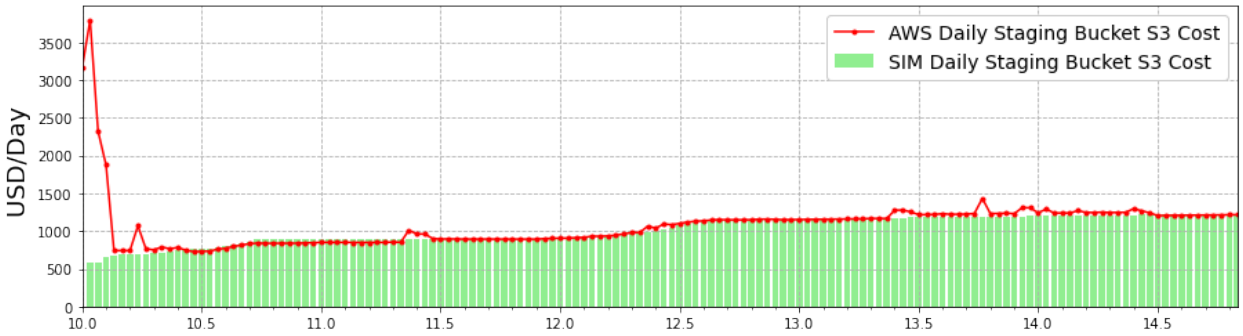


(c) AWS vs Sim Cumulative Cloud S3 Size (Model Ver.4)

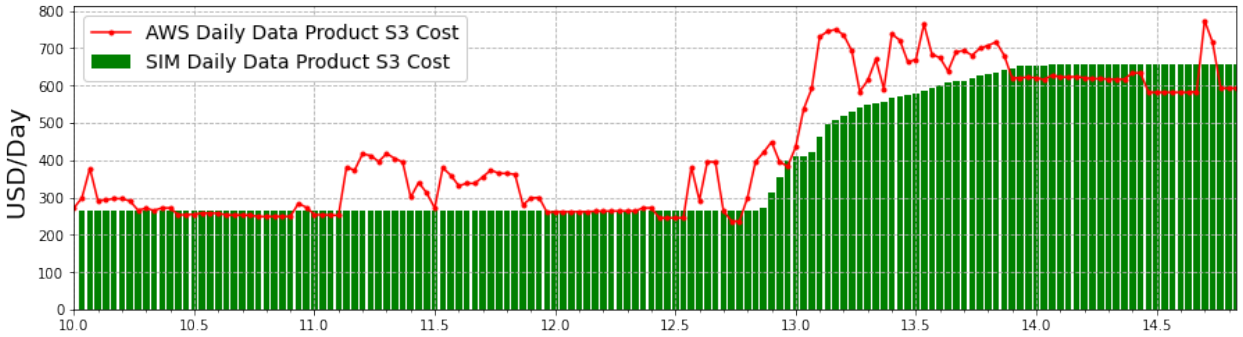
Figure 5.68: AWS vs Simulation Daily and Cumulative S3 Size Comparisons

5.5.2 *Dev-V&V* and *Operational Data System*

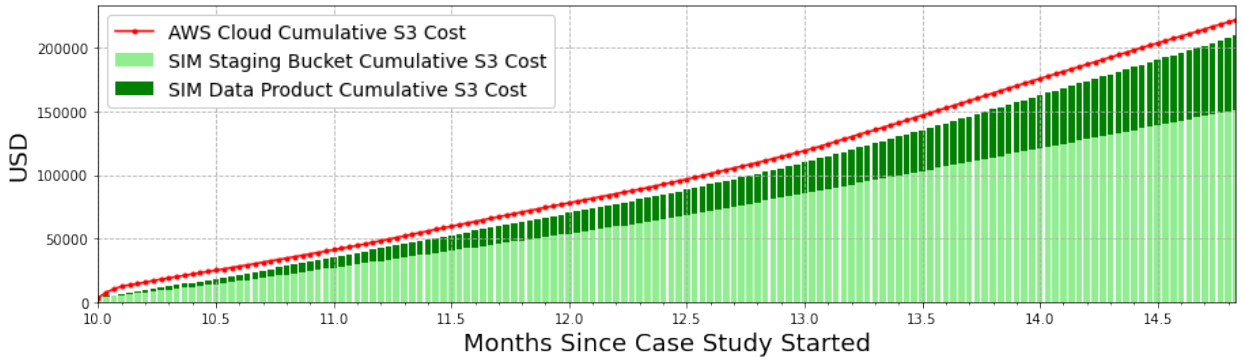
In this section, we provide an overview of the *Dev-V&V* and *Operational Data System* stages. This final YAMM iteration had very little software development and deployment activities. The data format 2 ETL pipeline software took the longest to finish developing. The ICD for this format had inaccuracies which led to many errors when parsing and translating the data. The final software



(a) AWS vs Sim Daily Staging Bucket S3 Costs (Model Ver.4)



(b) AWS vs Sim Daily Data Product S3 Costs (Model Ver.4)



(c) AWS vs Sim Cumulative Cloud S3 Cost (Model Ver.4)

Figure 5.69: AWS vs Sim Daily and Cumulative S3 Cost Comparisons (Model Ver.4)

version of this ETL pipeline was tested, delivered, and deployed to the operational data system at the beginning of this YAMM iteration.

As seen in the WBS schedule in Figure 5.5, the rest of the tasks in the four MVPs were completed in prior YAMM iterations. All other work was added to the WBS in the continuous YAMM

analysis and optimization phase for this prototype. Resource data was continuously collected during this phase as the end users continued to operate in their environment.

5.5.3 *Data Analytics & Algorithmics*

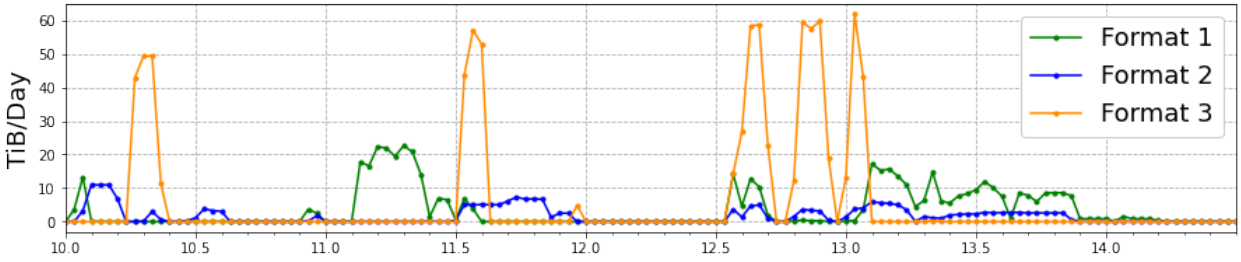
In this section, we discuss our work to analyze and extract latent variables, algorithms, constants, and other information for the cloud compute, networking, and service resources and costs. We also show the V&V input and validation data for the Ver.5 model.

We define the AWS terms and how they relate to each other for additional context throughout this section. A Virtual Private Cloud (VPC) is a logically isolated network within a public cloud infrastructure which allows organizations to create their own private, secure network environment. The Network Firewall (NetFw) is an AWS Firewall as a Service (FWaaS) that is designed to protect traffic within VPCs.

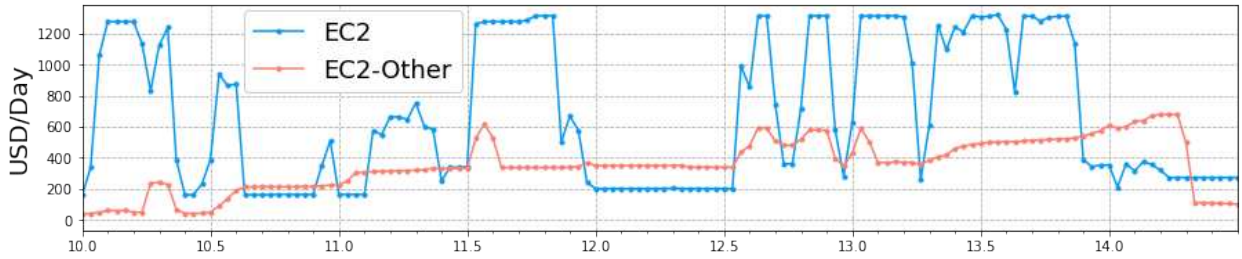
The AWS EC2 service offers various EC2 instance types with different compute, memory, storage, and networking capabilities. An EC2 instance is a virtual server which can have one to many "nodes" running on them. Nodes are analogous to VMs. The AWS EC2-Other Services (EC2-Other) refers to the other costs associated with using EC2 instances, but not the instance running itself. This is AWS's catch-all category for various additional services and resources related to each EC2 instance. Examples of EC2-Other costs include data transfer, EBS volumes and snapshots, Network Address Translation (NAT) gateways, elastic Internet Protocol (IP) addresses, and more.

Figure 5.70 visualizes each ETL pipeline's daily output (GiB) aligned with the four different AWS compute and network costs. Table 5.10 contains the correlation values we computed using a Pearson r correlation because we assumed that the mean daily cost and throughput resource data is normally distributed. The correlation values in this table show that the Format 3 ETL pipeline daily data throughput has a strong positive correlation to daily EC2 cost (0.780) over the full time frame. The full time frame of data we analyzed is from month 10 to month 14.5 since the start of the case study. This table also shows that format 3 daily throughput was the only ETL pipeline

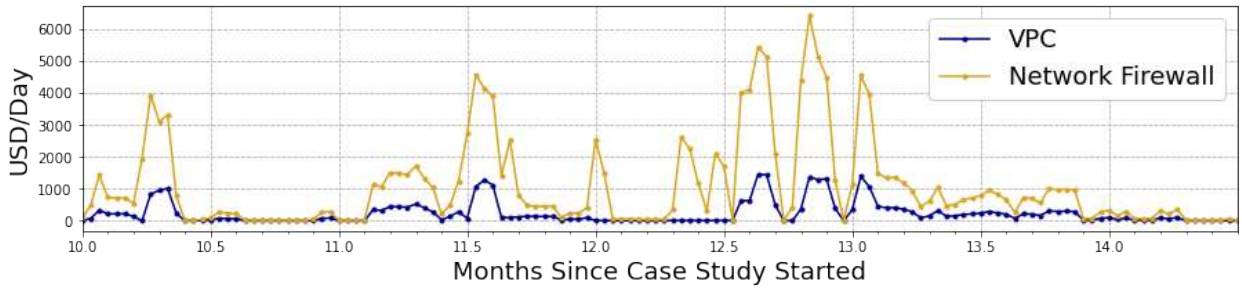
that had strong positive correlation to the VPC and NetFw daily costs. We saw in Figure 5.70 that there were windows of time which show that the other two formats' pipelines could have a strong positive correlation with compute and network costs when running in isolation.



(a) Daily Data ETL Throughput (Model Ver.5)



(b) Daily Compute Costs (Model Ver.5)



(c) Daily Network Costs (Model Ver.5)

Figure 5.70: ETL Data Pipelines Output Compared to Compute and Network Costs

Table 5.10 contains the correlation values for windows of time that we created to isolate discrete events to calculate each ETL pipeline's effect on compute and network costs. We called these time periods "correlation windows". The correlation windows are shown in units of months, which aligns with Figure 5.70. We created a correlation window for each of the three ETL formats when they were the only process running to calculate the variables we used to estimate cost from daily

pipeline throughput. The calculation of these pipeline variables is discussed in Section 5.5.3.1. Each pipeline and combinations of pipelines shows strong positive correlation to EC2 and VPC costs within their respective windows shown in Table 5.10 which lead us to believe these are the correct windows to use for the estimation of these variables. This table also shows that we did not find correlation windows that provided a strong positive correlation for each of the three pipelines to EC2-Other and NetFw costs. This resulted in using different methods to calculate these cost estimation variables and is also discussed in 5.5.3.1.

Table 5.10: Correlation of Daily ETL Throughput to Compute & Network Costs

ETL GiB/Day	EC2 ($/Day)	EC2-Other ($/Day)	VPC ($/Day)	NetFw ($/Day)
Correlation Window: Full Time Frame				
Format 1	0.382	0.148	0.305	0.190
Format 2	0.780	-0.03	0.371	0.292
Format 3	0.403	0.304	0.928	0.848
Format 1+2+3	0.601	0.321	0.996	0.874
Correlation Window: 11.1 to 11.4 Months				
Format 1	0.965	0.389	0.984	0.982
Correlation Window: 11.6 to 11.9 Months				
Format 2	0.940	-0.383	0.99978	0.125
Correlation Window: 10.2 to 10.5 Months				
Format 3	0.982	0.990	0.999	0.969
Correlation Window: 13.3 to 13.9 Months				
Format 1+2	0.698	0.288	0.849	0.829
Correlation Window: 12.8 to 13.1 Months				
Format 1+2+3	0.980	0.883	0.999	0.862

Table 5.10 shows that format 3 daily throughput had the strongest positive correlation with EC2-Other costs, but only in specific correlation windows. This is visualized in Figure 5.70 by the synchronized spikes of format 3 daily size and EC2-Other costs, however there was general upward trend in this cost not strongly correlated with daily throughput.

We extracted the EC2-Other cost breakdown from AWS, plotted them in Figure 5.71, and identified the cause of this upward trend in the cost data. We found that an old version of the Data Product data catalog was left active and did not elastically shut down unused processes which accounts for the general upward trend in EC2-Other costs. This old data catalog was shut down after month 14 and provided us with a new baseline data catalog cost used in the next section.

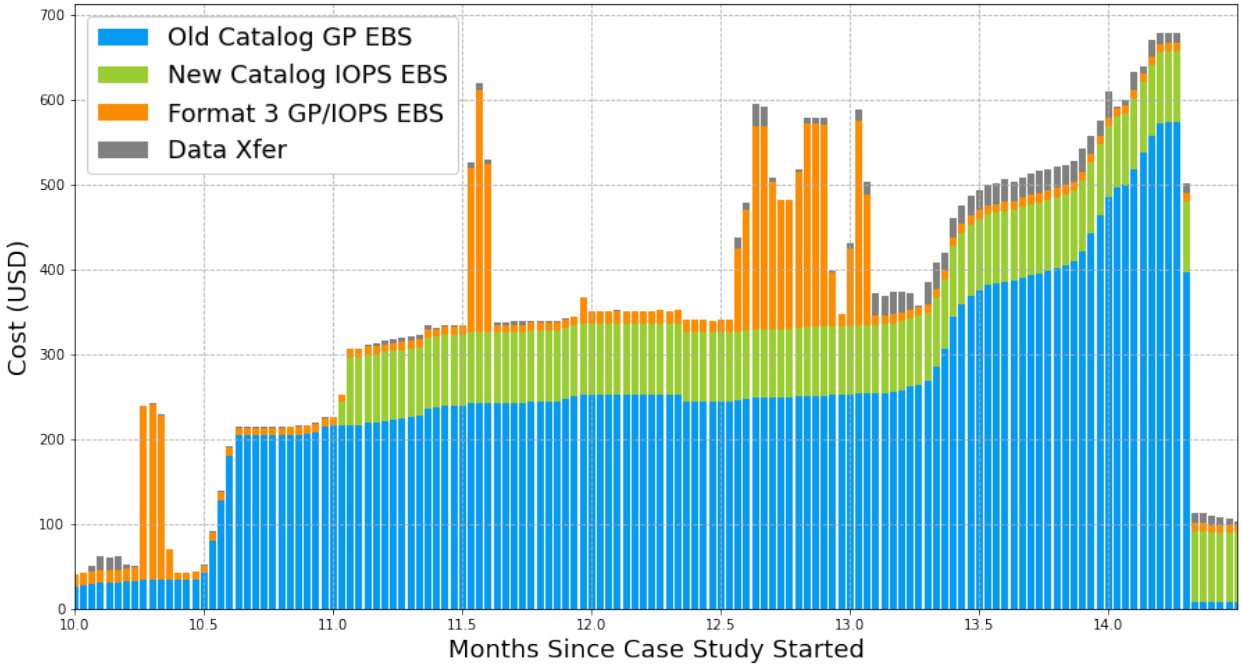


Figure 5.71: EC2-Other Cost Breakdown Time Series Data (Model Ver.5)

According to AWS documentation, there are several factors that can affect the performance and costs of EBS volumes, such as EC2 instance configuration, Input/Output (I/O) characteristics, and workload demand. The new data catalog transitioned from a General Purpose (GP) volume to a fully utilized Input/Output Operations Per Second (IOPS) provisioned volume which is shown in Figure 5.71 to be a cheaper alternative. Also shown in this figure was the contribution of format 3 pipeline's EBS cost to the EC2-Other costs aligned with the correlation windows in Table 5.10. The other pipelines did not require the same amount of EBS resources. The "other" costs in this figure are associated with EBS snapshots and data transfers.

5.5.3.1 Latent Variables & Algorithms

Table 5.11 contains the constants and latent variables derived from our cohort and factor analyses that we describe in this section. This table provides the units of these variables and a short description for context. These empirically derived variables enabled the Ver.5 model to estimate the compute costs, network costs, system resources, and processing performance. The variables in the table were calculated for the three data formats enable estimation on all of them.

Table 5.11: Variables to Estimate Cloud Data Processing Costs & Resources (Model Ver.5)

Variable	Units	Description
$DPC_{VPC}{}^{*}$	(USD/GiB)	VPC Data Processing Cost
$DPC_{EC2\text{-}Other}{}^{*}$	(USD/GiB)	Non-EC2 Data Processing Cost
$BC_{EC2}{}^{*}$	(USD/min)	EC2 Baseline Cost
$BC_{VPC}{}^{*}$	(USD/min)	VPC Baseline Cost
$BC_{EC2\text{-}Other}{}^{*}$	(USD/min)	EC2-Other Baseline Cost
$NetFw_{factor}{}^{*}$	-	NetFw Cost Factor
$EC2_{type}{}^{\#}$	-	AWS EC2 Instance Type
$EC2_{costRate}{}^{\#}$	(USD/EC2/min)	AWS EC2 Cost Rate by Type
$EC2_{qnty}$	(# of EC2s)	Number of EC2s Running
$EC2_{max}$	(# of EC2s)	Maximum Number of EC2s
$Pipe_{throughput}$	(Gbps)	Single ETL Pipeline Throughput
$Pipe_{perEC2}{}^{*}$	(pipelines/EC2)	Number of ETL Pipelines Per EC2
$DT_{min}{}^{+}$	(min)	Simulation Delta Time (Step)
DDS^{+}	(GiB)	Delta Data Size (DDS) per DT_{min}
$DDS_{pipe}{}^{+}$	(GiB)	Single ETL Pipe DDS
$DDS_{EC2}{}^{*+}$	(GiB)	Single EC2 Instance DDS
$DDS_{ETL}{}^{+}$	(GiB)	All ETL Pipes DDS

**Latent Variables*
$^{+}$*Simulation Variables*
$^{\#}$*System Metadata*

Shown in Figure 5.70b is a consistently flat upper EC2 cost limit which indicates the maximum number of EC2s set by the K8s resources. Additionally, there is a flat lower EC2 limit is indicative of a baseline cost of running K8s and other custom services that have allocated these resources. This upper and lower limit is the basis for the $EC2_{max}$ and BC_{EC2} variables in 5.11.

We calculate the data processing cost using (5.2), where $C_{processData}$ is the total cost of processing data, $C_{compute}$, is the compute cost component, and $C_{network}$ is the network cost component.

$$C_{processData} = C_{compute} + C_{network} \tag{5.2}$$

The compute and network cost components, calculated using (5.3) and (5.4), are each composed of two AWS categories of collecting resource and cost data for customers.

$$C_{compute} = C_{EC2} + C_{EC2\text{-}Other} \tag{5.3}$$

C_{EC2} is the cost of running the EC2 instances (the virtual servers) and $C_{EC2\text{-}Other}$ is the EC2-Other cost.

$$C_{network} = C_{VPC} + C_{NetFw} \tag{5.4}$$

We defined the Baseline Cost (BC) and Processing Cost (PC) as the 2 cost components for each of the four compute and network costs in (5.3) and (5.4). These four compute and network costs are defined in (5.5) through (5.7) in terms of the BC and PC. The BC represents the overhead cost of these resources when no data is being processed and, in contrast, the PC represents the cost of these resources while processing data. We assumed no data was being processed when the ETL daily throughput, shown in Figure 5.70a, was equal to zero.

$$C_{EC2} = BC_{EC2} + PC_{EC2} \tag{5.5}$$

$$C_{EC2\text{-}Other} = BC_{EC2\text{-}Other} + PC_{EC2\text{-}Other} \tag{5.6}$$

$$C_{VPC} = BC_{VPC} + PC_{VPC} \tag{5.7}$$

$$C_{NetFw} = C_{VPC} \cdot NetFw_{factor} \tag{5.8}$$

The NetFw cost (C_{NetFw}) in (5.8) is calculated using an empirically derived factor multiplied by the VPC cost. We used this factor because the VPC cost had a strong positive correlation (0.882) to the NetFw cost over the full time frame.

We estimate the BCs in (5.6) and (5.7) for each data format as the mean cost of these resources while no data was being processed. EC2-Other costs, shown in Figure 5.71, had an issue with the old data catalog resource de-allocation which resulted in the steady rise in cost. We chose to set the EC2-Other BC to the value on the far right that only contains the new data catalog so future scenario simulations represent the latest data catalog costs.

To calculate the BC for each simulation time step, we multiplied each BC constant by DT_{min}. The BC in 5.5 is discussed in the next section. The estimation of the PC for each of the three compute and networks costs was more complex and is described in more detail in the next section also.

5.5.3.2 Network & Compute Costs

This section describes the work to parameterize the three PC compute and network costs in (5.5) through (5.7), estimate the latent variables, and build the algorithms that were integrated into the Ver.5 model. We used the the *Data Analytics & Algorithmics* process illustrated in Figure 3.20 and described in Section 3.4 to accomplish this work.

Before the discussion about the analyses we performed it is necessary to provide context about the Ver.5 model components and assumptions used for the cohort and factor analyses. The cloud native ETL pipelines created for the hybrid cloud prototype are complex. They use a pipeline orchestration tool which spins off many different processes to parse, translate, and load the data into the data catalog and datastore. These are a mixture of parallel and serial processes that are all executing within a pod or container within a K8s cluster. The K8s cluster manages the nodes (VMs) that these pods are running on and the EC2 instance that the nodes are on. Additionally, the communication between nodes for the ETL pipeline, the S3 staging bucket, the S3 datastore, and the VM hosted data catalog requires VPC, NetFw, and EC2-Other resources.

We parameterized this complex ETL pipeline orchestration and K8s resource management subsystem as a single abstract data pipeline to simplify the UM and reduce the analytics needed to quantify it. These single ETL data pipelines can run in parallel while K8s balances resources and runs new process on nodes and EC2 instances that have resources remaining. Therefore, each EC2 instance can have one to many single ETL pipelines running on it.

For this parameterized single abstract data pipeline we developed the ETL_{perEC2} latent variable which is the number of pipelines per EC2 instance. This is an empirically derived number that depends on the data format, software processes, pipeline orchestration tool, K8s configuration, EC2 instance type, EBS type, S3 type, VPC setup, NetFw setup, and more. These multiple layers of complexity and rapid software refactors are the reason we parameterize this complex behavior and why we must use empirically derived variables.

We now move on to define more latent variables to parameterize UM components. We started with a cohort analysis using the model components to identify matching cohorts in the resource data. From this, we developed (5.9) which use values the model already calculates. Then, we moved to a factor analysis and identified latent variables to calculate $EC2_{qnty}$ using (5.10). Lastly, we defined a decision tree algorithm within the Ver.5 model which is represented by the function in (5.11).

$$PC_{EC2} = EC2_{qnty} \cdot DT_{min} \cdot EC2_{costRate} \tag{5.9}$$

$$EC2_{qnty} = ceil(\frac{DDS_{ETL}}{DDS_{pipe} \cdot Pipe_{perEC2}}) \tag{5.10}$$

$$DDS_{ETL} = f(Pipe_{throughput}, Pipe_{perEC2}, EC2_{max}, Pipe_{queue}, Pipe_{interval}, Pipe_{trigger}) \tag{5.11}$$

The PC_{EC2} in (5.9) is the EC2 PC from (5.5). The Ver.4 model already splits the simulation into discrete time steps (DT_{min}), calculates the DDS of the entire ETL pipeline (DDS_{ETL}). We define the DDS as the amount of data, in GiB, that flows into, out of, or through an object during the simulation's discrete time step (DT_{min}). The $EC2_{costRate}$ is a value AWS posts for each EC2 instance type. The rest of the variables in these equations, DDS_{pipe} and $Pipe_{perEC2}$ are empirically derived from the resource data.

The parameterizations for the PC variables in (5.6) and (5.7) were simpler to derive than the one for the EC2 PC that we just described. These parameterizations in (5.12) and (5.13) are proportional to the Data Processing Cost (DPC) latent variable which was derived from the strong positive correlation between the size of the data being processed and the VPC and EC2-Other costs. The VPC and EC2-Other DPCs for each of the three data formats were empirically derived from the data.

$$PC_{EC2\text{-}Other} = DPC_{EC2\text{-}Other} \cdot DDS_{ETL} \tag{5.12}$$

$$PC_{VPC} = DPC_{VPC} \cdot DDS_{ETL} \tag{5.13}$$

The BC in 5.5 was calculated using 5.14 which uses the EC2 instance type running the K8s management services, and estimate for number of EC2s, and the simulation time step.

$$BC_{EC2} = EC2_{qnty} \cdot EC2_{costRate} \cdot DT_{min} \tag{5.14}$$

The equations above were provided to the SE team to integrate into the Ver.5 UM. We did this modeling work in parallel with this data analysis to ensure the latent variables leveraged current model capabilities which reduced analysis and model integration time.

5.5.3.3 V&V Data

This section describes the V&V input and validation data that we created for the UMA and sent to the SE team to implement the last *MDAO* stage. The only V&V time series input data we needed is shown in Figure 5.70a. This data is used as a way to control when each the three ETL pipelines are executing for every software version that ran in that environment.

We adjusted storage costs and sizes in a similar manner as we did for the in Section 5.4.4.2 for model validation. This iteration we limited the adjustments to the large manually deleted data, as before, and adjusted the initial Data Product storage size to zero. For simplicity, we assumed the staging bucket already contained all the data since our main goal was to validate the new added compute and network cost estimation capabilities. These adjustments to the S3 data sizes are shown

in Figure 5.72 in units of Tibibytes (TiB). Figure 5.73 shows these corresponding adjustments to the S3 costs that will be used to validate the Ver.5 model.

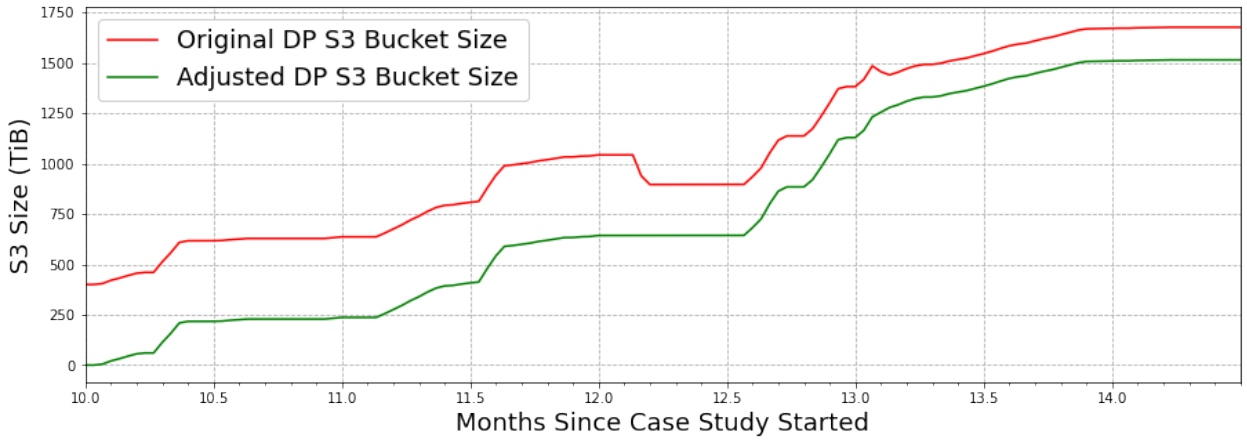


Figure 5.72: Data Product Datastore S3 Bucket Size Adjustments (Model Ver.5)

Figure 5.73: Data Product Datastore S3 Bucket Cost Adjustments (Model Ver.5)

The validation of the EC2 compute time series costs is shown in Figure 5.70b. In addition to the storage sizes and costs, we adjusted the EC2-Other validation time series costs by removing the old catalog EBS costs once the new catalog was deployed. We also removed the data transfer costs shown in Figure 5.71. We didn't want an old version of the data catalog affecting future

optimizations so we didn't model this behavior. We also didn't model any data transfer behavior in the Ver.5 model so we removed that from the validation data as well.

The validation data we used for the time series network costs is shown in Figure 5.70c. We did not adjust any of these costs. We also collected an average AWS enterprise monthly cost for input into the Ver.5 model's recurring cost capability that is used for the system optimization.

Table 5.12 contains the empirically derived latent variables and inputs that were passed to the SE team for their UM simulations. This table shows the correlation factors for each ETL data format. It also shows the assignment of the BC cost variables to a K8s component rather than assigning it to the ETL pipelines which more accurately reflected reality. Additionally, this table shows the K8s EC2 variables which are used to calculate the BC_{EC2} cost in (5.14).

Table 5.12: Variables to Estimate Cloud Data Processing Costs & Resources (Model Ver.5)

Variable	Format 1	Format 2	Format 3	K8s
$DPC_{VPC}{}^{*}$	0.02004	0.01921	0.01936	0
$DPC_{EC2\text{-}Other}{}^{*}$	0	0	0.00361	0
$BC_{VPC}{}^{*}$	-	-	-	0.00595
$BC_{EC2\text{-}Other}{}^{*}$	-	-	-	0.07835
$NetFw_{factor}{}^{*}$	-	-	-	4.0
$EC2_{type}$	C6in.16XL	C6in.16XL	C6in.16XL	M5.XL
$EC2_{costRate}$	0.06084	0.06084	0.06084	0.0032
$EC2_{qnty}$	-	-	-	45
$Pipe_{throughput}$	0.15621	0.1457	0.14967	-
$Pipe_{perEC2}{}^{*}$	5	1	13	

5.6 YAMM: Continuous Analysis & Optimization

In this section we describe the final system analysis and optimization completed during this case study. The last of the software and DI were deployed in the previous iteration, so now we move onto the continuous analysis and optimization phase of YAMM. Prior to this phase, the software has been matured, the operational environment has been running, large amounts of resource data has been collected and analyzed, and the UM with it's libraries have been validated. In this phase

we focused on maturing the capabilities of the UM to estimate more than the basic storage sizes and costs.

In the life cycle of a production system this section aligns with the start of the data system sustainment phase. This sustainment phase is discussed in the Continuous YAMM Analysis and Optimization in Stage 5 in YAMM Process Flow 1. This stage focuses on iterative full system optimizations with higher fidelity models rather than focusing on single MVPs or low fidelity system models. In Section 5.6.1 we describe the full implementation of our *MDAO* stage applied to the Ver.5 model. For this prototype system we present the capabilities of the latest updated Ver.5 UM and the results of the model validation. We also present the trade study setup, the trade study objective functions, and the trade study inputs towards Pareto-Optimal solutions and system analysis.

5.6.1 *MDAO*

This section provides details of our Ver.5 UMA model update, model validation, system optimization using three objective functions, and the Pareto, trade space, and sensitivity analyses from the optimization results. Section 5.6.1.1 describes the Ver.5 UM compute and network cost and resource estimation updates we made to the model for the cloud and the on-prem subsystem components. Section 5.6.2 provides the results of the UM V&V in terms of quantitative measurements and time series plots.

Section 5.6.3 describes the two scenarios we defined that represent near term and long term use case possibilities as the data system begins to scale in data collection and processing. This section includes the three objective functions and decision variables we defined for the system optimization stage. Section 5.7 shows the results of the system optimization stage. In Section 5.7.3 we show the results of our Pareto, trade space, and sensitivity analyses. This provides insight into the dominant system characteristics and trade space clusters that present in both near term and long term scenarios.

5.6.1.1 UMA Ver.5 Model Upgrade

In this section we describe the compute, network, and recurring cost estimation capabilities we added to the Ver.5 UM for each of the three data formats. The compute and network resource and cost algorithms added to the model were based on the work done in Section 5.5.3. That section identified the equations, latent variables, and constants to calculate the compute and network costs from the daily processed data size (GiB) for each of the three ETL pipeline formats. This section also describes the capabilities we added to the model to account for initial purchase costs, replacement costs, and recurring or service costs for cloud or on-prem subsystem components.

The capabilities we added to this model were focused towards our specific objectives for this data system and towards system optimization and analysis. In addition to optimizing costs and performance, we also defined the following trade study objectives for this prototype:

1 Identify the best recording format and communication protocols between the three formats

2 Identify the optimal storage utilization strategy between EBS and S3

3 Define the optimal data retention and cloud governance policies

4 Estimate on-prem CapEx and OpEx costs for the hybrid cloud system

The extensive updates we made to the Ver.5 model to meet these trade study objectives followed the same top-down meets bottom-up model development pattern we described in the UMA in Section 3.1.1.2. In this section we start the description of the updates from the top-down. We started with updating the definition of this case study's primary use case to include the use case needed to validate the model and the use case that will be used to run the trade study for the system optimization and analysis. These two use cases included in the the primary use case shown in Figure 5.74. Updating the cloud Data Product with multiple data formats via local server is the validation use case, while the other will be used for the trade study. The validation use case and the trade study use case each include their own sub-use case, but also include one they share as seen in the figure.

Figures 5.75 and 5.76 show the IBDs of the physical hybrid cloud prototype and Data Product we used for the validation of the Ver.5 model. Figure 5.75 shows several upgrades to the hybrid

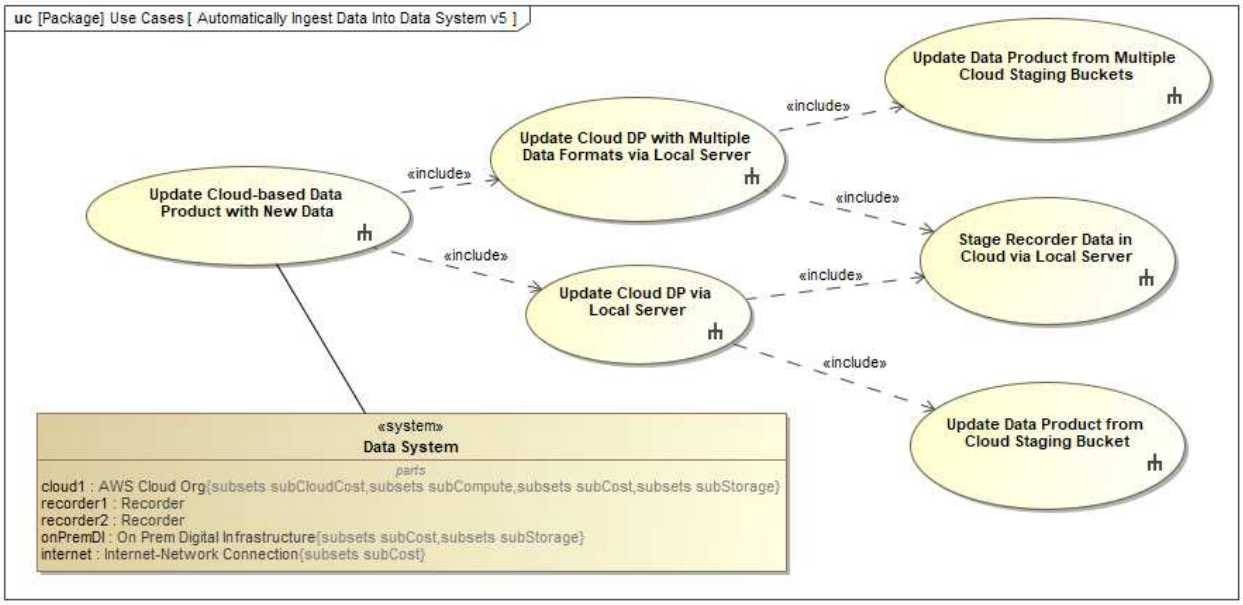


Figure 5.74: Updated Case Study Primary Use Cases (Model Ver.5)

cloud data system compared to the previous version shown in Figure 5.14. These upgrades included a second recorder, an on-prem DI, the internet, multiple staging buckets, a data archive service, and an archive storage bucket.

The second recorder represents a second OLTP, or aircraft, system feeding data into the hybrid cloud data system for use in trade studies. The on-prem DI was added to estimate the CapEx and OpEx expenses of the local sub-system. The Daqscribes and Snowballs were still present in this on-prem DI, however they were not used for this validation so were left out of this analysis model. The internet block was added to the model to limit the internet throughput speeds and to add purchase and recurring internet costs to the total system cost. The data archive service and associated archive storage was added to the model to help us analyze optimal data retention policies.

The three staging buckets and data pipelines shown in Figures 5.75 and 5.76, respectively, were added to validate the compute and networking cost and resource estimations for the three prototype data formats. We discuss this validation in more detail in Section 5.6.2. These separate staging buckets and ETL pipelines were not included in the simulations for the system optimization and

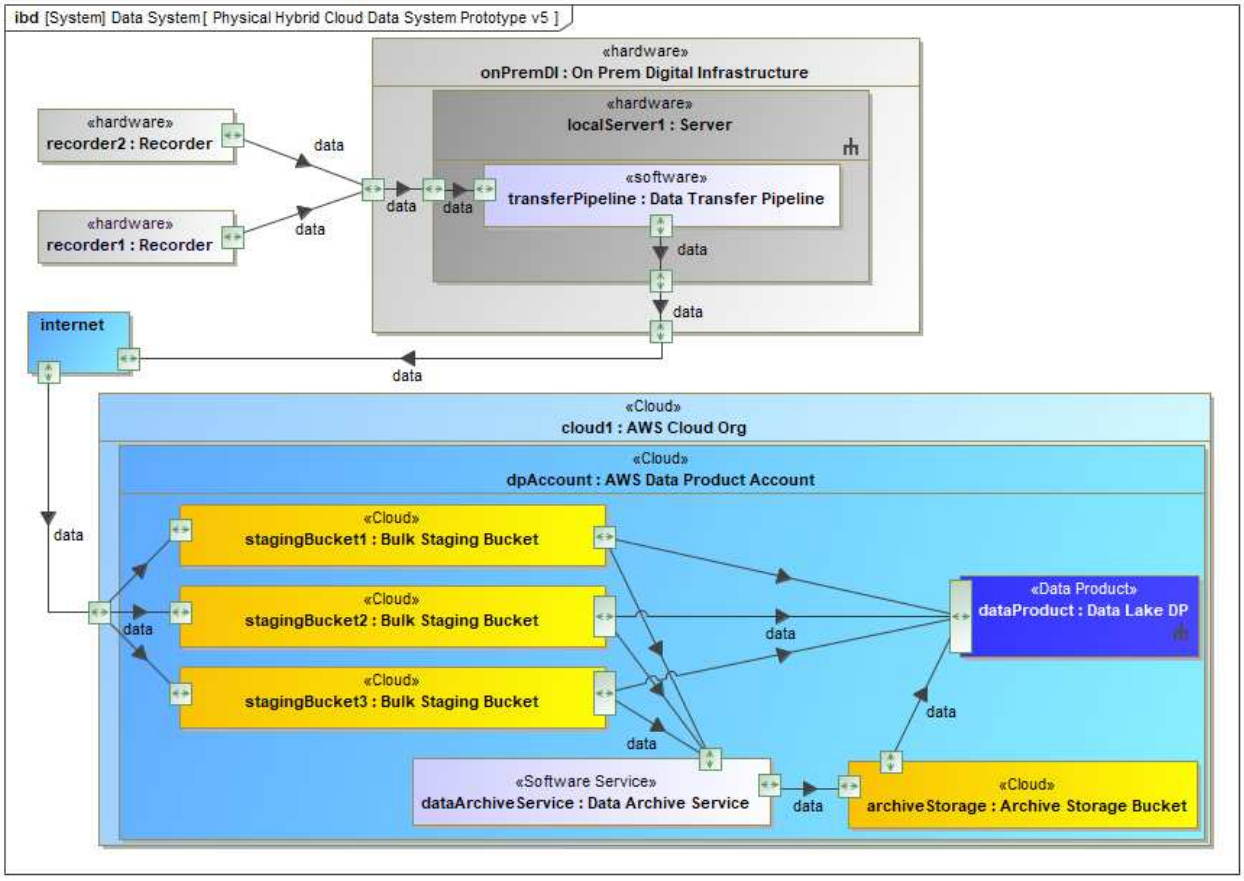


Figure 5.75: Updated Hybrid Cloud Data System with Multiple Formats for Validation (Model Ver.5)

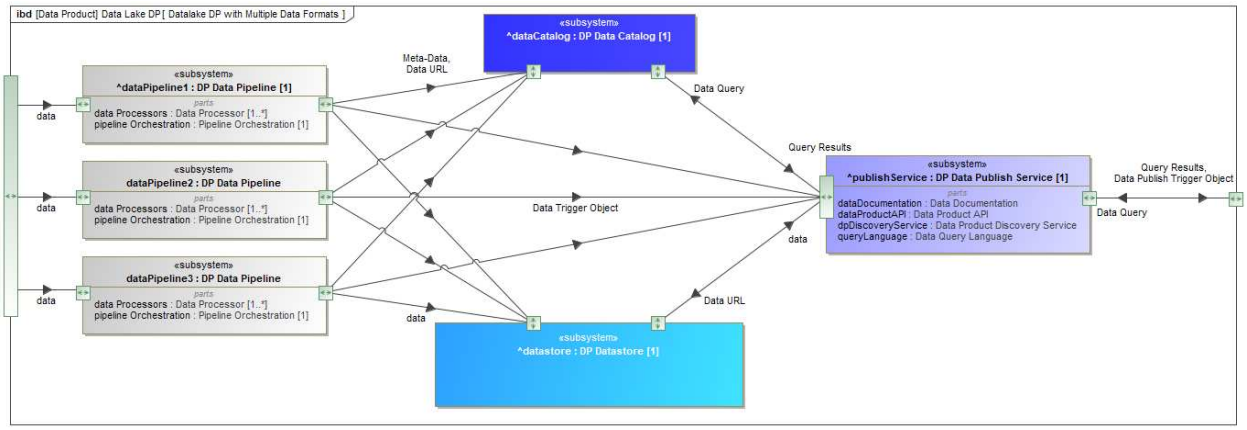


Figure 5.76: Updated Datalake Data Product with Three ETL Pipelines for Validation (Model Ver.5)

analysis. We built a simulation matrix to analyze each data format separately to meet trade study objectives 1 and 2 above. We discuss this strategy more in Section 5.6.3.

Figure 5.77 is the activity diagram for the validation use case which updates the Data Product from the three staging buckets. This figure shows two behaviors which perform ETL and data archive for these buckets. During the operation of the AWS prototype there were no data archive services running so the archive behavior in this figure is not during validation. This data archive behavior was built to help define the data retention policy for objective 3 and is discussed more in Section 5.6.2.

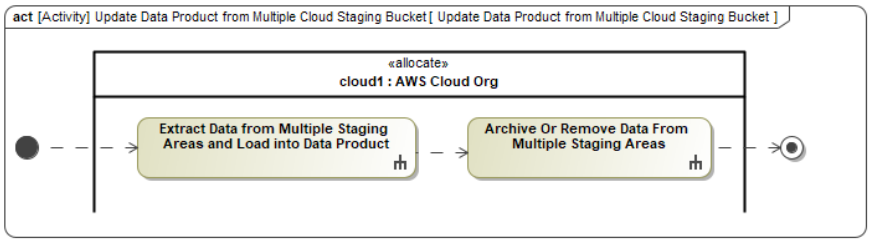


Figure 5.77: Update Data Product from Multiple Staging Buckets (Model Ver.5)

Figure 5.78 shows the activity diagram of the multiple behaviors that define each ETL pipeline. Figure 5.79 shows the specification behavior of the ETL process that updates the Data Product for data format 1. This behavior is the same for data formats 2 & 3 with the exception that the behaviors in the first two columns of the allocation swim lanes are allocated to the respective staging bucket and data pipeline parts. This behavior was an update to the ETL behavior shown in Figure 5.35 where the majority of the updates were done on the visual simulation and function layers of the UM libraries.

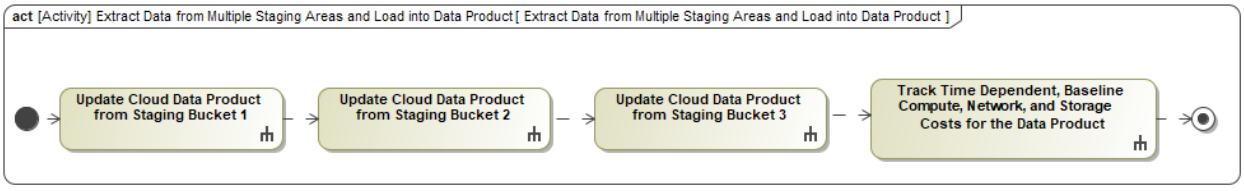


Figure 5.78: Updated Activity for Multiple ETL Pipelines into a Single Data Product (Model Ver.5)

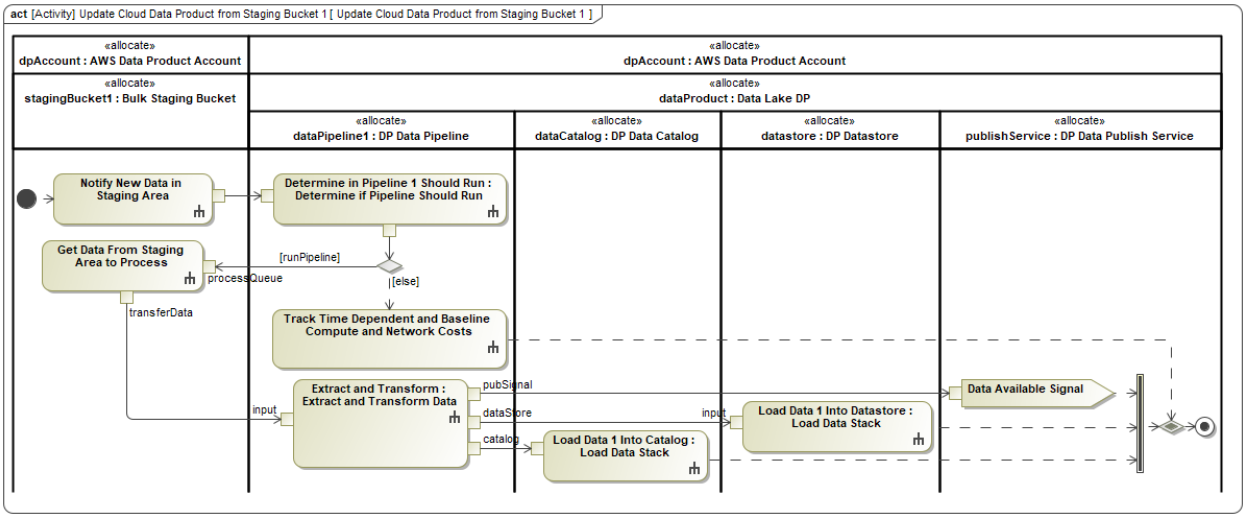


Figure 5.79: Activity for ETL Pipeline 1 into the Data Product (Model Ver.5)

Figure 5.80 shows the value property and constraint updates we made to the abstract objects to enable the compute, network, recurring, and purchase cost estimations within the model. These include value property rollups for the new resource and cost values. We also split out baseline costs from cloud costs to reduce the number of value properties assigned to on-prem sub-system components. We did this to increase the speed and reduce the memory requirements of the simulation execution. Each object in the data system inherits from one to many of these abstract objects so the amount of memory required increases exponentially when we add value properties. We discuss the performance results of these simulations in more detail in Section 5.6.3.

5.6.2 Unified Model Validation Results

This section presents our results of the V&V of the Ver.5 UM in terms of our UMA quantitative accuracy measurements, AWS vs simulation estimate time series plots comparisons, and observations about root cause of errors. Table 5.13 contains the quantitative accuracy measurements for the simulation's estimations of S3 size and cost, compute costs, network costs, and total costs for the Data Product. We also included the total costs for AWS and the simulation to provide extra context. Max entropy for the time series residuals is also shown at the bottom of the table.

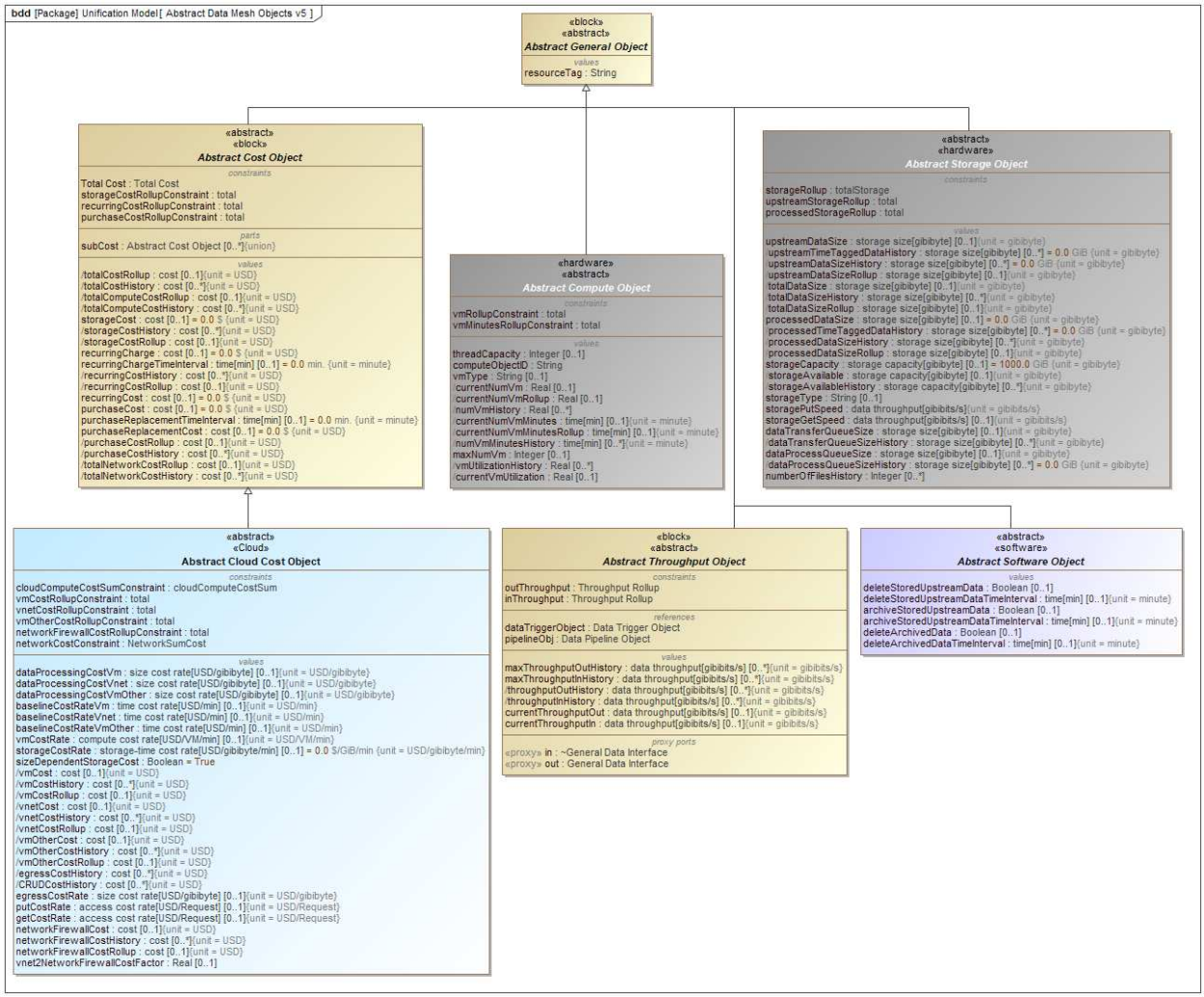


Figure 5.80: Abstract Data Mesh Objects (Model Ver.5)

Figure 5.81 compares the AWS empirical data and simulation estimates of the daily ETL data processed loaded into the datastore. These values are expected to be very similar since the time series data V&V input was daily ETL throughput. Table 5.13 results for S3 size reflect this accuracy in terms of one of the lowest percent errors and a RMSE more than 4 OOMs lower than the total S3 size. However, the residual entropy is nearly the lowest of values in the table. Again, we want a larger residual entropy number that approaches the maximum possible entropy, which is the maximum possible random noise possible using (3.3). This combination of high accuracy in percent error and RMSE and low accuracy in residual entropy indicates that we are missing a small non-random behavior or a scalar variable is off slightly.

Table 5.13: Data Product Quantitative Measurements of Accuracy (Model Ver.5)

Variable	Percent Error	RMSE	Residual Entropy	Max Entropy Percent Error	AWS Total	Sim Total
Data Product S3 Storage:						
S3 Size:	1.0%	215.3 GiB	2.8	38.0%	1,531 TiB	1,516 TiB
S3 Cost:	9.8%	$90.00	3.5	24.0%	$75,839.57	$68,437.75
Data Product Compute Costs						
EC2:	18.5%	$258.32	3.3	29.0%	$91,068.40	$74,249.19
EC2-Other:	4.5%	$47.20	2.9	37.0%	$17,436.01	$18,228.31
Total:	14.8%	$253.35	3.4	27.0%	$108,504.41	$92,477.50
Cloud Network Costs						
VPC:	1.0%	$32.44	3.0	36.0%	$31,842.10	$31,509.90
NetFw:	26.1%	$737.47	2.4	49.0%	$145,068.70	$107,133.65
Total:	21.6%	$747.14	2.4	47.0%	$176,910.80	$138,643.55
Storage, Compute, & Network:						
Total Cost:	17.1%	$864.60	3.1	33.0%	$361,254.79	$299,558.80
Max Entropy: 4.6						

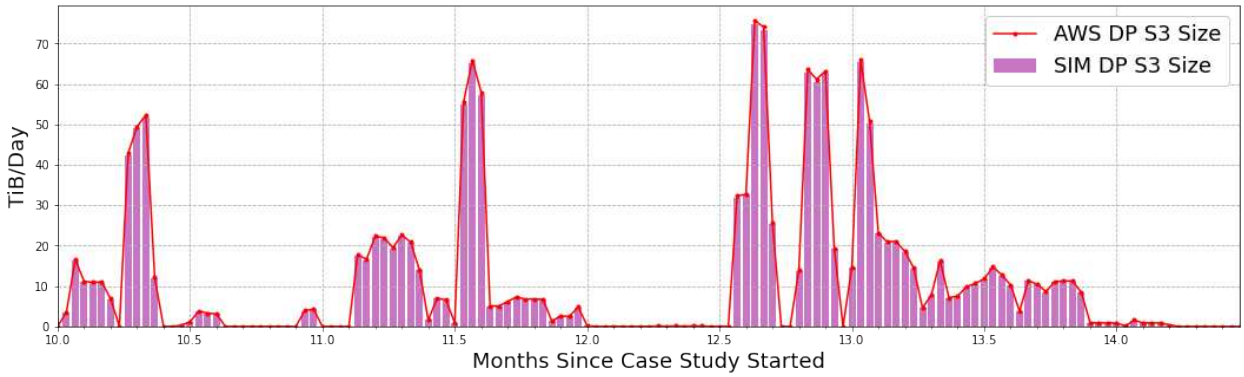


Figure 5.81: Sim Vs. Real Data Product S3 Size Comparisons (Model Ver.5)

Figure 5.82 shows the daily S3 cost comparison between the AWS resource data and the Ver.5 simulation estimates. These results show that the AWS daily costs include more cost spikes than the smoother simulation S3 cost estimates. This is also reflected in Table 5.13 that shows the a higher percent error than the S3 storage size and the RMSE is only 3 OOMs lower than the AWS total price. While this is still a good RMSE it is an OOM larger than the S3 size RMSE. Interestingly, the residual entropy of the S3 cost is higher (better) than the S3 size. This indicates that the residual is more stochastic, however when we saw these results we noticed these costs spikes appear to

visually align with the format 1 & 2 data processing times. We found a strong positive correlation of 0.889 from the format 1 & 2 daily S3 sizes, which indicates that we missed modeling a behavior associated with processing data. After research we found that S3 CRUD charges are included with the S3 costs and formats 1 & 2 use a significant amount of CRUD actions.

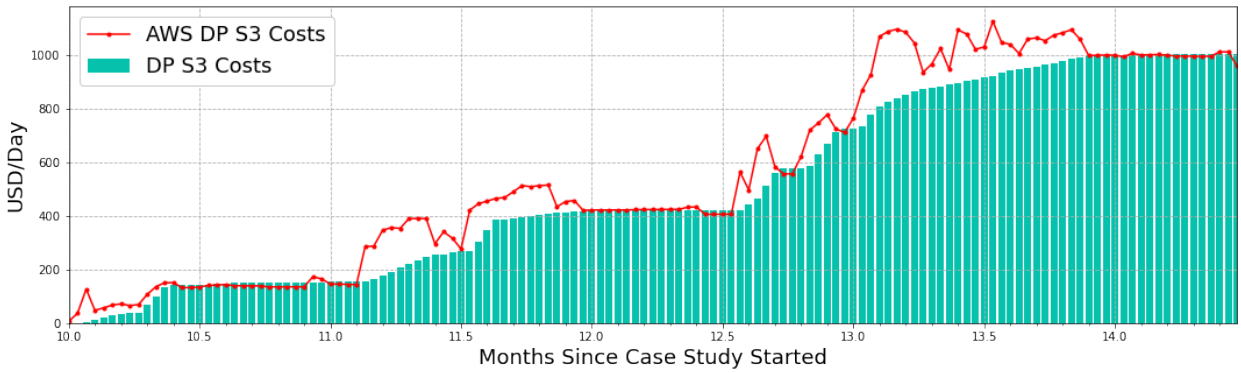


Figure 5.82: Sim Vs. Real Data Product S3 Cost Comparisons (Model Ver.5)

Figure 5.83 shows the EC2 cost comparisons and also shows the same visual correlation of formats 1 & 2 processing magnitude line up with the S3 cost spikes in Figure 5.82. Figure 5.84 shows the EC2-Other cost comparisons of AWS and the simulation, but also shows the lack of any costs associated with formats 1 & 2. These results align with the storage utilization strategy that formats 1 & 2 ETL pipelines performed CRUD actions on the the S3 objects directly, the while format 3 ETL performed I/O on EBS volumes instead. These EBS costs show up in Figure 5.84. These observations are what led to our objective 2 above that we will answer with the system optimization and analysis in Sections 5.7 and 5.7.3.

Figure 5.84 shows the large spike in the AWS EC2-Other data between month 10.5 and 11. This spike is the old data catalog EBS costs before the new data catalog was deployed. We based our simulation baseline on the newest data catalog baseline costs and by the results we are over predicting this slightly. The percent error and RMSE for EC2-Other are good values, however the residual entropy value is low. These quantitative measurements, like the ones for S3 size, indicate

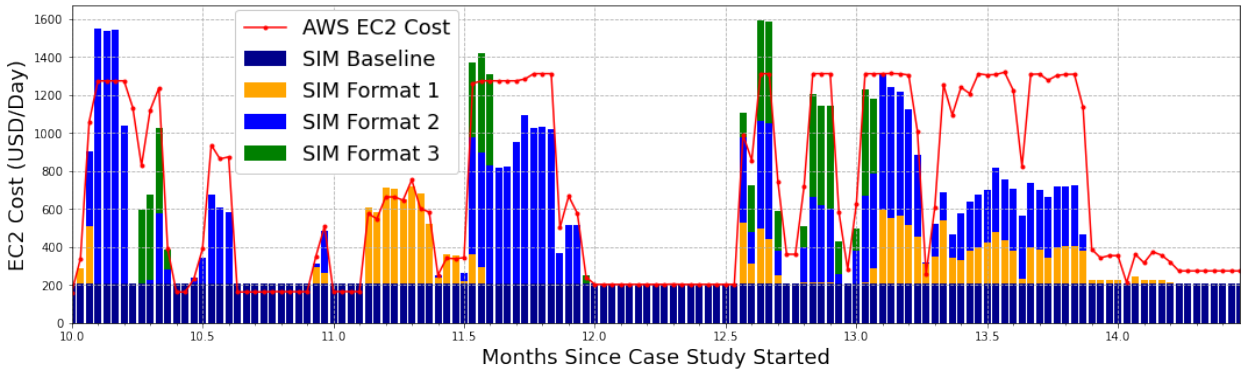


Figure 5.83: Sim Vs. Real Data Product EC2 Cost Comparisons (Model Ver.5)

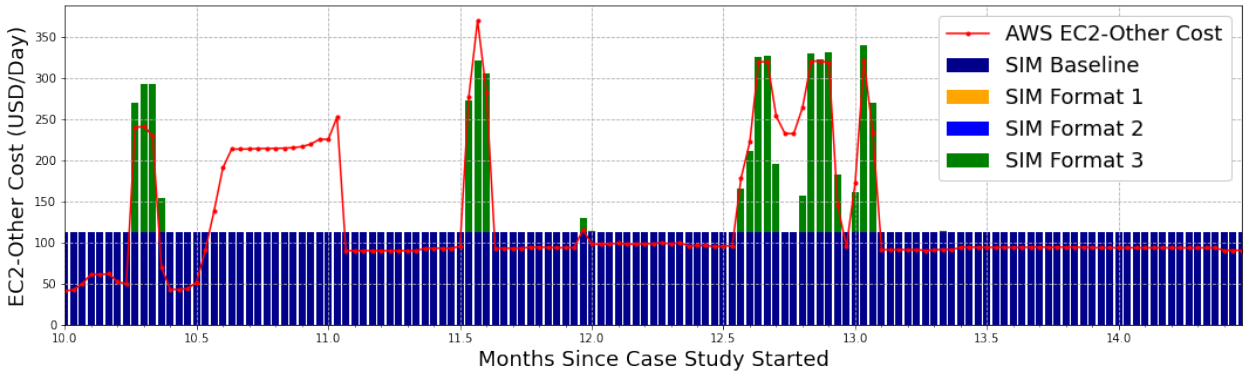


Figure 5.84: Sim Vs. Real Data Product EC2-Other Cost Comparisons (Model Ver.5)

that we are close to the overall cost estimates, but we have missed a small non-random behavior or a scalar variable is off slightly.

Figure 5.85 shows the time series VPC comparisons between the actual costs and the simulation estimations. These time series plots match very well and Table 5.13 shows that these simulation VPC cost estimations are the most accurate in terms of percent error. The RMSE is 3 OOMs lower which is also accurate, but the residual entropy shows that there are non-random residuals that we may have missed.

Figure 5.86 shows the comparison between the time series simulation estimations and actual costs of the NetFw. This figure shows several large cost spikes our estimations did not predict. Table 5.13 shows that the percent error for this cost is the worst of all other predictions. The RMSE is close to 2 OOMs lower than the total price which is the worst as well. Finally, the

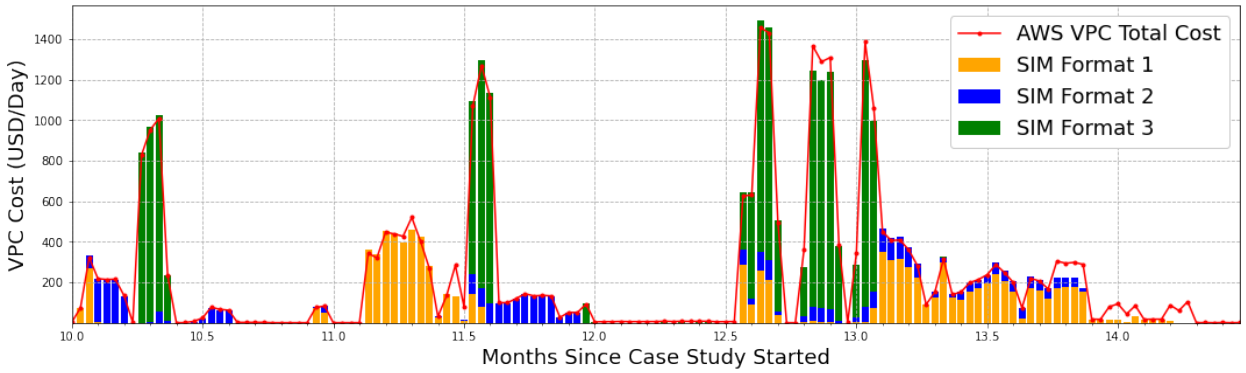


Figure 5.85: Sim Vs. Real Data Product VPC Cost Comparisons (Model Ver.5)

residual entropy is also the lowest which means our estimations for NetFw are the weakest of all cost estimations.

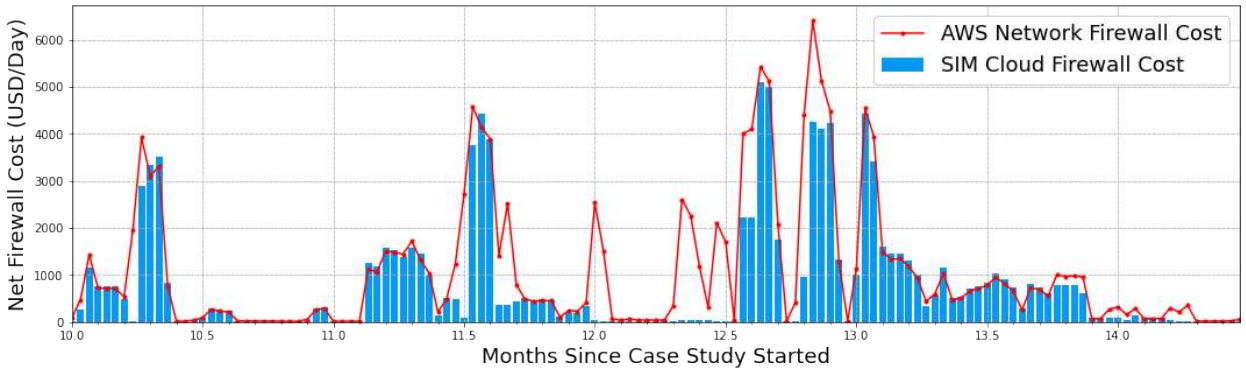


Figure 5.86: Sim Vs. Real Data Product NetFw Cost Comparisons (Model Ver.5)

The NetFw service costs are reported at the AWS cloud tenant level, whereas the VPC costs are reported at the account level. The results above show that we are accurately estimating the VPC costs, therefore the spikes seen in Figure 5.86 are most likely attributed to other accounts within our tenant organization.

Table 5.14 shows network cost estimations from the AWS website compared to our simulation estimations and the actual costs. This table shows that our simulation cost estimations are better than the AWS website estimations. The network cost estimations from the website appear simple and are listed as a factor of a flat hourly cost and a price per GiB. However, we are unsure if our

organization got a discounted price or if we are calculating the website estimations incorrectly. Our compression rate for data was roughly 50%, so did not know when to apply the uncompressed data size or the compressed data size for the estimations. We used the uncompressed data for the NetFw calculation since it was clearly going between two accounts, then used the compressed size for the VPC estimation since it was internal to our account. Either way, these website estimations were not more accurate than our simulation.

Table 5.14: AWS Website vs Simulation vs Actual AWS Costs (Model Ver.5)

Estimation Method	AWS Website	UM Simulation	AWS Actual	Website Percent Error	Sim Percent Error
VPC:					
Hourly Rate Total:	$145.80	-			
Compressed GiB Rate Total:	$70,563.40	-			
Total cost:	$70,709.20	$31,509.9	$31,842.1	122.1%	1.0%
Network Firewall:					
Hourly Rate Total:	$1,279.80	-			
Uncompressed GiB Rate Total:	$203,849.81	-			
Total Cost:	$205,129.61	$107,133.65	$145,068.7	41.4%	26.1%
Total Network:					
Total cost:	$238,101.81	$138,643.55	$176,910.8	34.6%	21.6%

Finally, Figure 5.87 shows the time series simulation estimations of storage, network, and compute cost totals compared to the actual AWS costs. This figure represents all of the data and cost estimates contained in the precious figures leading up to this one.

For context, the total cost percent error in Table 5.13 is roughly the same percent error as the average cloud budget overrun from the Flexera report [12], however the largest errors in total cost stem from the NetFw which are most likely due to activity outside of the account we were predicting. Therefore we have specific areas of improvement for our simulation because it has been broken into simplified components. These areas of improvement were identified throughput this section because we broke the simulation into simple components and had a range of quantitative measurements for each estimation which pointed to the issue.

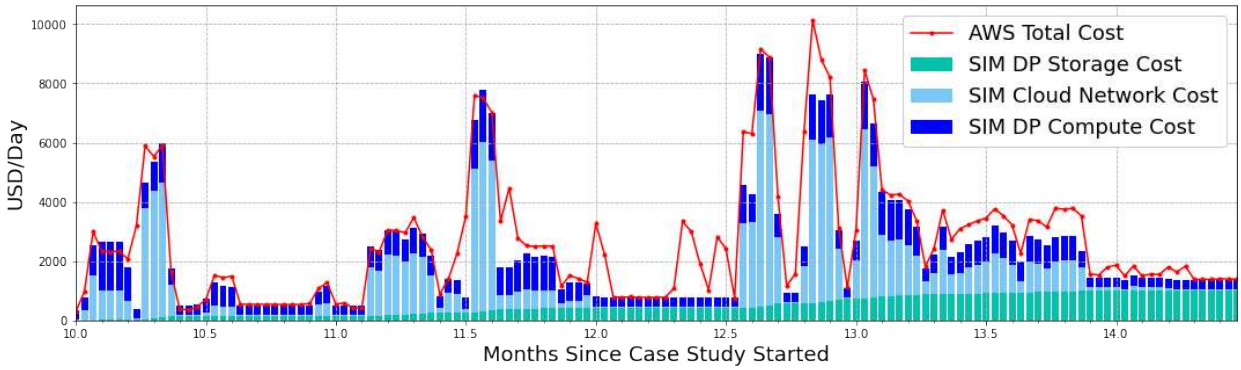


Figure 5.87: Sim Vs. Real Data Product Total Cost Comparisons (Model Ver.5)

The results in this section show that our Ver.5 model has captured the majority of the behaviors and the ones not captured have simple solutions. These results validate that our model is working correctly even though we have additional behaviors to model and possibly scalar variables to adjust. Even with some of it's inaccuracies this model is ready to complete the rest of the *MDAO* stage and move on to system optimization and analysis.

5.6.3 Trade Study Setup

In this section we outline the trade study we defined which encompasses a new model use case and simplified structure, two scenarios to compare, 3 objective functions to optimize, the decision variables involved. We also summarize the on-prem CapEx and OpEx estimations in this section as well as the speed of simulation execution. We defined this trade study to meet our trade study objectives 1 through 4 we discussed in 5.6.1.1. We also collect data for our research question about the performance of our UM.

Figure 5.88 shows the simplified structure we used for this trade study. The on-prem DI includes a single local server and a single staging bucket and data format. We built the model input files to specify which data format to use during the simulation to analyze the optimal format for future requirements. This is how we answered trade study objectives 1 and 2. This figure also shows the data archive service and archive storage capability we added to the model. The IBD of the Data Product in this figure is still defined in Figure 5.15.

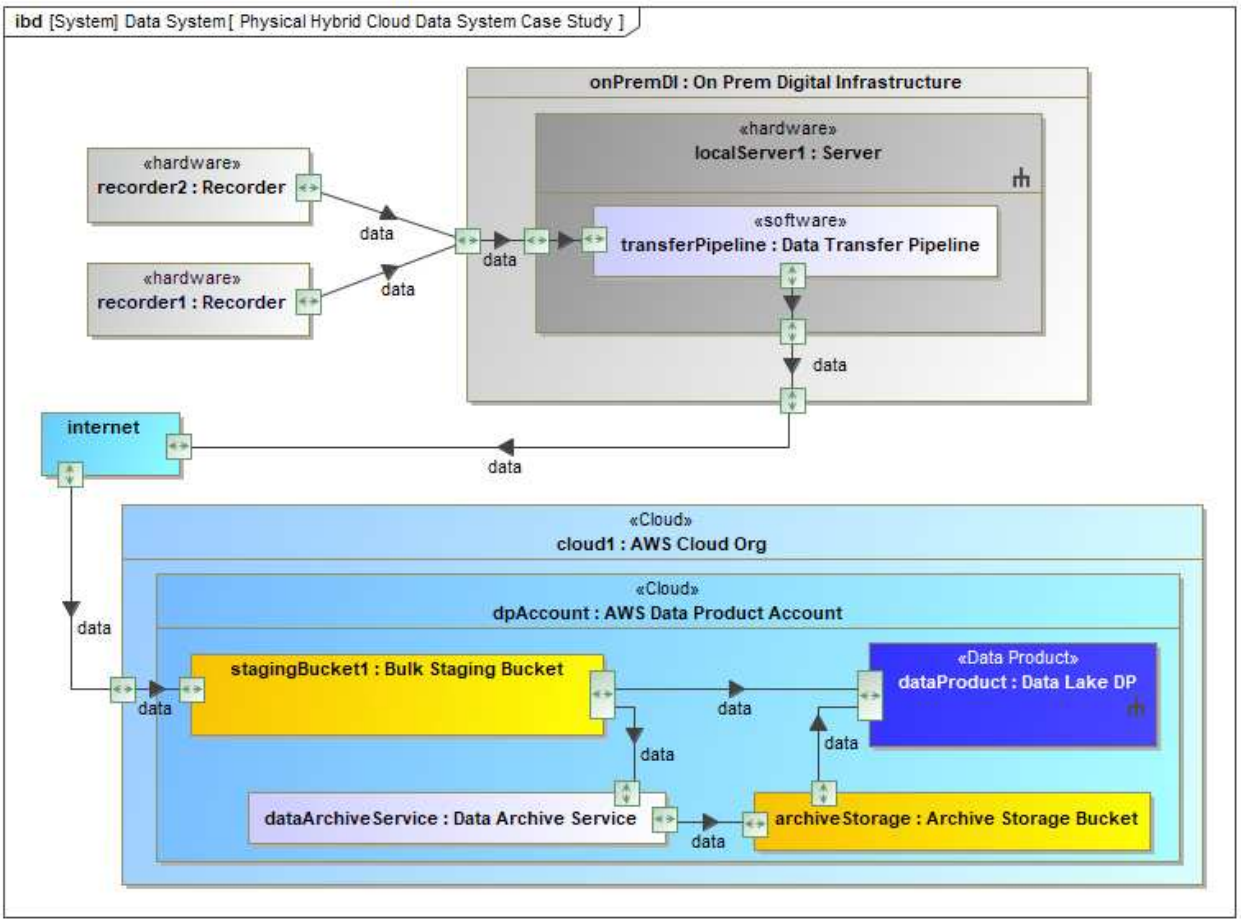


Figure 5.88: Hybrid Cloud Data System for Optimization and Analysis (Model Ver.5)

We defined this trade study with different scenarios resulting in the need to download data from the two recorders to the on-prem server, then automatically upload this data to the cloud. This behavior is part of the "Update Cloud DP via Local Server" use case which was created for this trade study. This trade study use case is shown in Figure 5.74 and includes two other use cases. The first use case for staging recorder data in the cloud is further defined in the activity diagram in Figure 5.89. This figure shows the visual specification viewpoint of the capabilities we added to Ver.5 to download data from two recorders and allocate a behavior to the internet component which we used to keep track of internet data throughput history at the simulation level.

Figure 5.90 shows the activity diagram of the second use case included the trade study use case shown in Figure 5.74. This activity diagram contains two activities allocated to the cloud component. The first one is further defined in the activity diagram in Figure 5.91 which specifies the

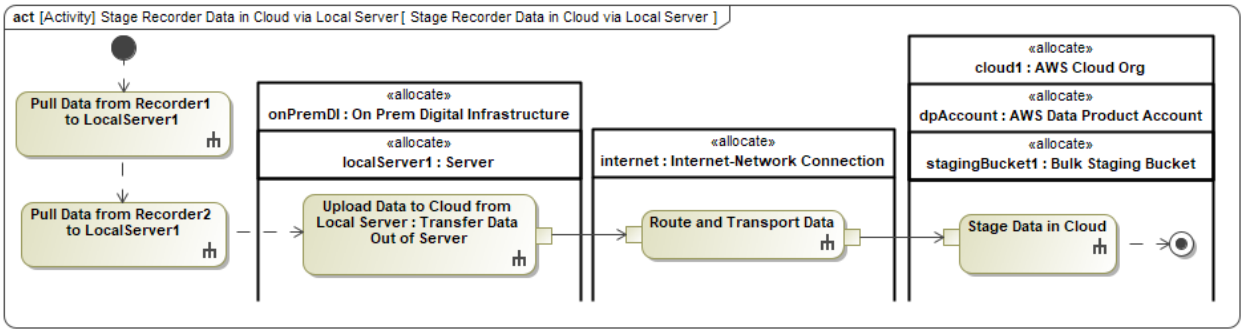


Figure 5.89: Uploading Both Recorder's Data into the Cloud for Optimization and Analysis (Model Ver.5)

behavior of the ETL pipeline. The second activity defines the data archive service that implements our data retention policy via several decision variables we define later in this section. The data archive service activity diagram is shown in Figure 5.92.

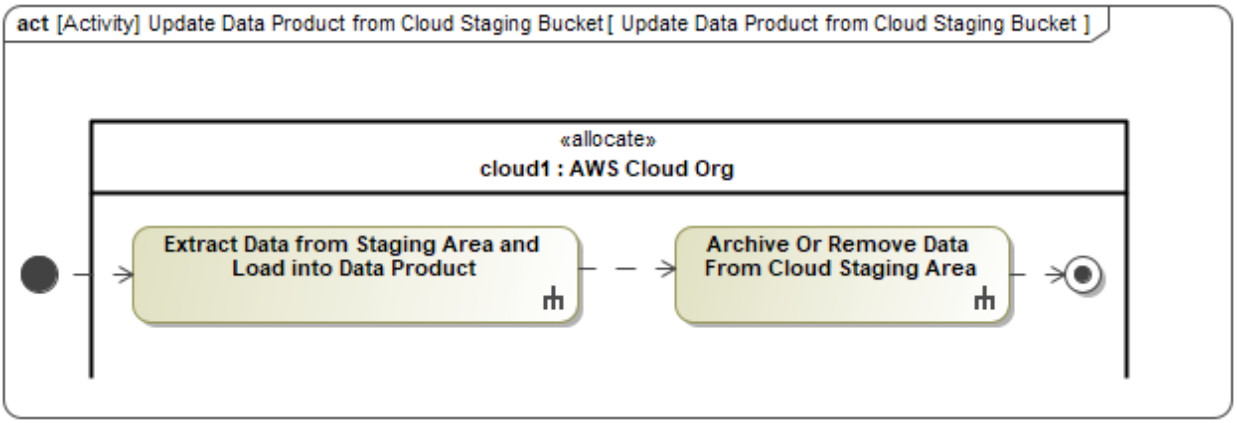


Figure 5.90: Updating Data Product from a Staging Bucket and Running the Archive Service(Model Ver.5)

Both Figures 5.91 and 5.92 show new activities in these visual specification viewpoints that enable tracking time dependent compute, storage, and networking resources and costs. These activities call visual simulation behaviors and simulation functions and utilities to estimate the continuous estimations of the resources and costs.

5.6.3.1 Trade Study Objective Functions:

We defined 3 objective functions for this trade study to address objectives 1 through 3. Our multi-objective problem formulation is shown in (5.15) with the constraints in (5.16). Our goals is

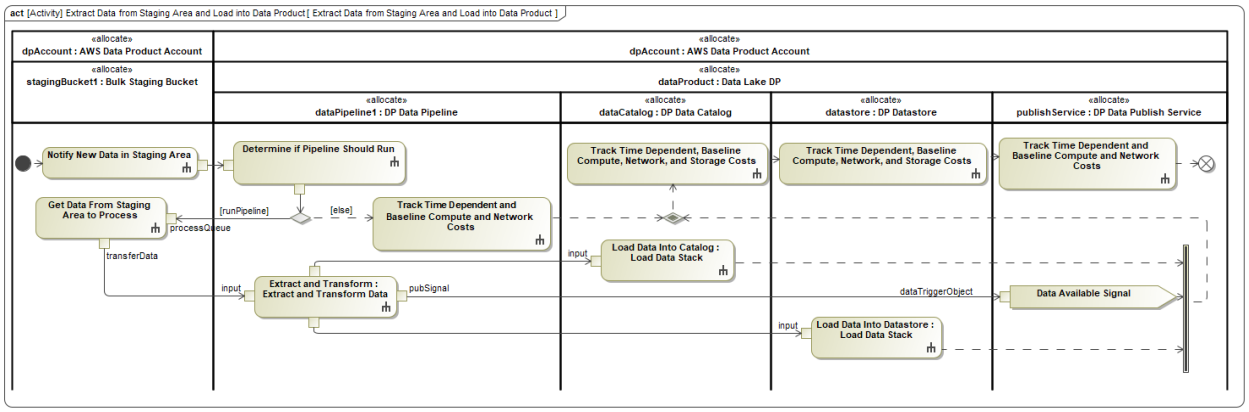


Figure 5.91: ETL Pipeline Behavior for Optimization and Analysis (Model Ver.5)

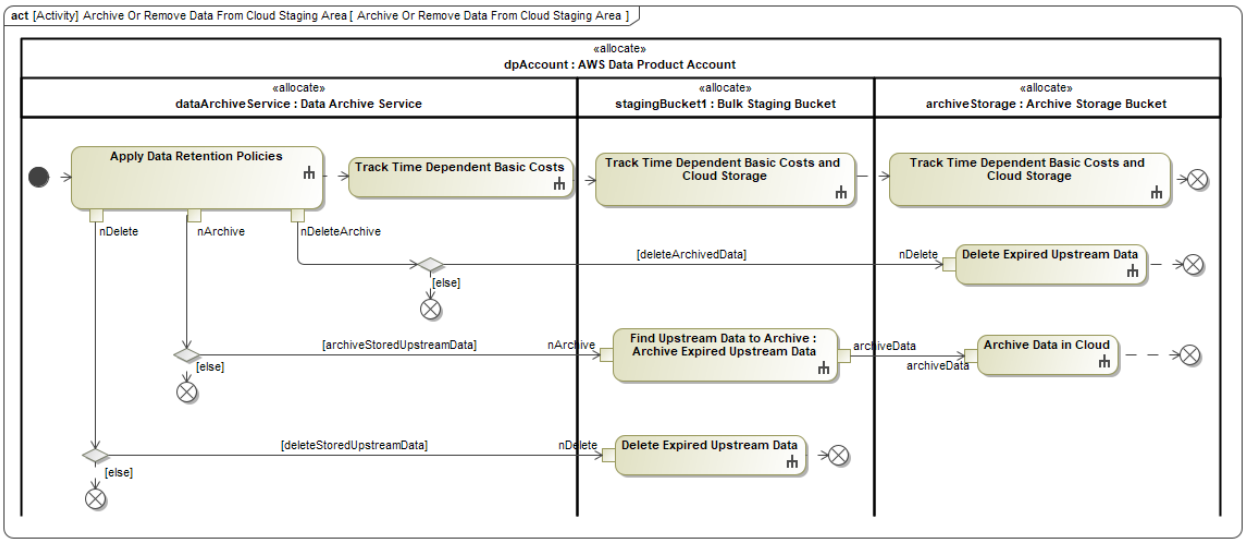


Figure 5.92: Cloud Archive Service Behavior for Optimization and Analysis (Model Ver.5)

to maximize objective functions f_{perf} and f_{TVD} and minimize f_{cost}. Each function in (5.15) has a dependency on $\mathbf{x}$ and t, but can only be solved through the use of a simulation. We also used the Min-Max scaling defined by (4.7) when calculating each Objective Function Values (OFVs) during the system optimization to prevent large number biases.

$$\underset{\mathbf{x}}{\textbf{minimize}} : f(\mathbf{x}, t) = \begin{bmatrix} f_{cost}(\mathbf{x}, t) \\ -f_{perf}(\mathbf{x}, t) \\ -f_{TVD}(\mathbf{x}, t) \end{bmatrix} \tag{5.15}$$

$$\textbf{subject to:} \quad \begin{cases} \{x_1 \in [1,2,3]\} & (Data\ Format) \\ \{x_2 \in \mathbb{R} \mid 0 \leq x_2 \leq 100\} & (Internet\ Throughput\ (Gbps)) \\ \{x_3 \in \mathbb{N} \mid 1 \leq x_3 \leq 20\} & (Maximum\ EC2s) \\ \{x_4 \in \mathbb{N} \mid x_4 \geq 0\} & (Days\ Until\ Archived) \\ \{x_5 \in \mathbb{N} \mid x_5 \geq 30\} & (Archival\ Retention\ Days) \\ \{t \in \mathbb{N} \mid t \geq 0\} & (Time) \end{cases} \tag{5.16}$$

Where the cost function is defined by (5.17). C_{cloud} is the total cost of the cloud and $C_{internet}$ is the total cost of the internet. We left out the on-prem costs since they were static for this trade study.

$$f_{cost}(\mathbf{x}, t) = C_{cloud}(\mathbf{x}, t) + C_{internet}(x_2, t) \tag{5.17}$$

The performance objective function shown in (5.18).

$$f_{perf}(\mathbf{x}, t) = \left(\frac{1}{n \cdot S_{max}} \sum_{t=0}^{n} S_{size}(\mathbf{x}, t) \right) \tag{5.18}$$

Where $S_{size}(t)$ is the S3 storage size at time step t, n is the total number of time steps (days) in the simulation, and S_{max} is the max size possible of the Data Product datastore. We divide by these values to create a non-dimensional performance function value.

Finally, the Time Value of Data (TVD) objective function is defined in (5.19) to represent raw recorded data's value after it has already been parsed and translated. For example, during this prototype we saw several instances where the ICD or the parsing and translation processing scripts were incorrect, so the raw recorded data had to be processed again. If this raw data had been deleted immediately after it was processed this data would have been lost and therefore there is some value in keeping this data around for some time. However, this data is not worth keeping for years since most bugs have been solved within the first several months of deploying the final

product. To calculate the TVD we had to create a new set of algorithms in the UM to tag data by the date it is uploaded to the cloud or archived. This time tagged storage data also used with the archive service algorithm we created to delete or archive data that had been stored past its retention date.

The TVD in (5.19) is the sum of the weighted standard storage (SS_{size}) and the weighted archive storage size (SA_{size}) over time from the calculation date to the date in the past where the TVD equals zero (t_{TVD_o}). Government policy, local regulations, or the organization itself determines the t_{TVD_o}.

$$f_{TVD}(\mathbf{x}, t) = \sum_{t=0}^{-t_{TVD_o}} W_{TVD}(t) \cdot (SS_{size}(\mathbf{x}, t) + W_A \cdot SA_{size}(\mathbf{x}, t)) \tag{5.19}$$

$SS_{size}(t)$ is the standard storage size at time step t, SA_{size} is the archive storage size, or AWS Glacier, at time step t, and W_A is the archive storage weight. We applied an archive storage weight (W_A) of 0.90 since archive storage can hae a slower retrieval time. For instance, AWS Glacier storage has retrieval times from 1 minute to 12 hours. Therefore we assign the archive storage less value than standard storage. We applied a smoothed TVD step function ($W_{TVD}(t)$) which is defined in (5.20) for each previous time step.

$$W_{TVD}(t) = 6x^5 - 15x^4 + 10x^3, \left\{ x = \left(\frac{t}{t_{TVD_o}} + 1 \right) \right\} \tag{5.20}$$

An example of the TVD weight function with a t_{TVD_o} of 180 days is shown in Figure 5.93.

5.6.3.2 Trade Study Inputs:

Before we created a simulation matrix or simply ran a PSO algorithm on the UM simulation, we analyzed the simulation execution speed as a function of time steps. This was done to estimate the time to complete the trade study analysis. Figure 5.94 shows the results of the execution time for the Ver.5 V&V model simulation. This figure shows an quadratic increase in execution time for the number of time steps. At 300 time steps the simulation times out due to insufficient RAM and

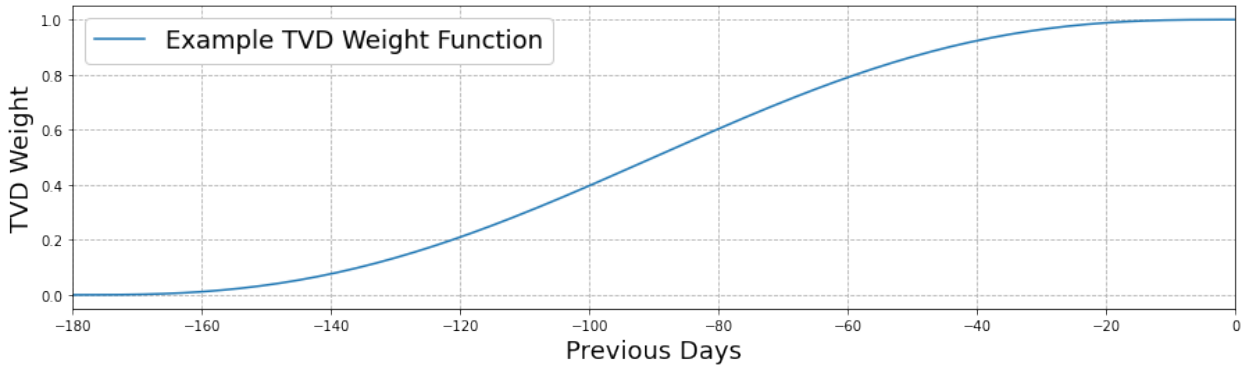


Figure 5.93: TVD Smoothed Step Weight Function ($t_{TVD_o} = 180 Days$)

memory swapping. In addition to this, the VM that the TWC server-side simulation was running on had issues and the new VM's performance was drastically reduced resulting in a 300% increase in execution time. This slower execution time is shown in this figure. This issues was not fixed before running the rest of the simulations. The smaller optimization simulation use cases are also shown in the figure. These show that the execution time has a dependency on the internet speeds. This is a function of the model logic that does not run the processing estimation algorithms if there is no data in the processing queue. The simulations with faster internet speeds process the data faster and reduce the execution speed.

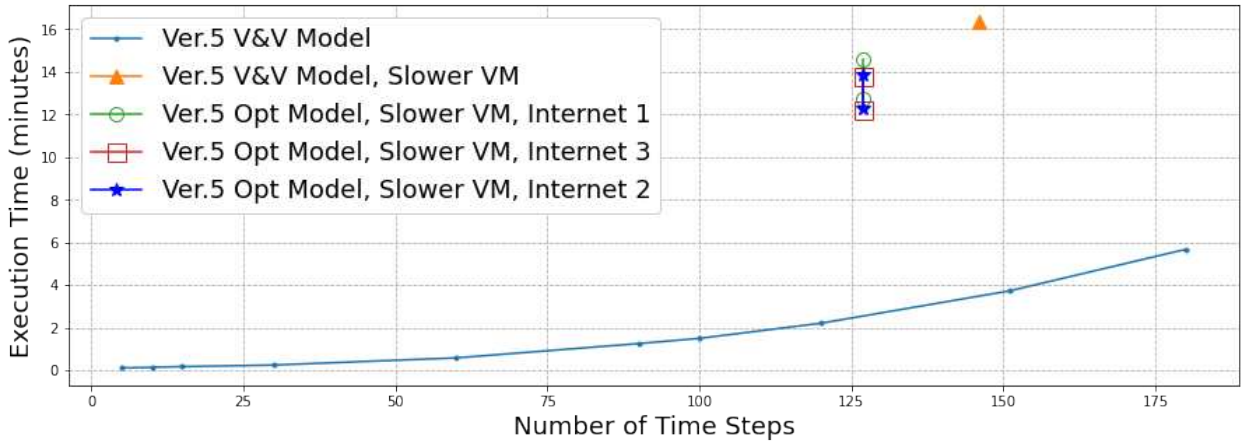


Figure 5.94: Simulation Execution Time by Number of Time Steps

Table 5.15 shows the decision variables and their domains we defined to meet our trade study objectives 1 through 3. This table shows the decision variables' discrete domains we defined for this trade study within the constraints of (5.16). The data format and internet packages were naturally discrete variables. However, we defined the other three decision variables' domains in the table as discrete to limit the number of simulations we had to run. We limited these domains to reduce the total simulation execution time to produce a set of Pareto optimal solutions for analysis. The PSO algorithm turned out to converge too slowly on this model when using the slower server-side VMs. Therefore we chose to run a simulation matrix using the limited decision variables and domains to produce a Pareto analysis. This analysis will define the dominant system characteristics and allow us to focus our future simulations and optimizations on a smaller domain.

Table 5.15: System Optimization Decision Variables (Model Ver.5)

Decision Variable	$(\mathbf{x}, t)$	**Discrete Domain**
Data Format	x_1	[1, 2, 3]
Internet Package	x_2	[1, 2, 3]
Max ETL EC2 Instances	x_3	[5, 10, 20]
Days until Archived	x_4	[30, 60, Never]
Archival Retention Days	x_5	[30, 60, Forever]
Time (Days)	t	126 Days

Table 5.15 show the three data formats used in this hybrid cloud prototype so we analyzed the performance and costs of these separately to determine which recording format and communication protocol pair was optimal for the data system. The three internet packages are listed in Table 5.16. We defined the maximum EC2 instances that the cloud ETL data pipelines were allowed to use since we saw that this made a difference cost and performance in previous analyses.

To determine a good data retention policy, we defined a decision variable to vary the number of days data sits in the staging bucket until it is archived. One option is to never archive the data, the other two options of 30 and 60 days were chosen so that we could see the effects of this decision

variable easily within the 18 week simulation time frame. The last decision variable defines the amount of time to delete data after it has been archived. Again, one option is to never delete data, while the other options were chosen to ensure that the effects could be visible within 18 weeks. The maximum option of days to archive and days to delete is 120 days which is roughly 17 weeks. While it is typical to archive data as soon as possible, it would not be typical to delete data so quickly. This is just a limitation of the simulation time frame, but we were still able to analyze the differences in the OFV to determine what our dominate characteristics were.

For the trade study we created a simulation matrix which contained every combination of decision variables in Table 5.15 with the exception that we ignored archival retention days of 30 & 60 for combinations if the data was never archived. This simulation matrix resulted in 378 simulation combinations. For context, the PSO algorithm took 145 simulations to converge on a single set of weight coefficients for all three objective functions. If we chose to adjust these coefficients like we did in Section 5.4.1.4, then we would have had to converge 1,000 more PSOs resulting in roughly 145,000 simulations which would have taken too long.

In Table 5.16 we defined the throughput and price of the three internet packages that connect the on-prem DI to the cloud. These were based off local commercial internet in our area. Additionally, we defined the average daily throughput to be stochastic within a normal distribution to better represent reality. We based the standard deviation values on the internet connection upload speeds from the prototype. This table also shows the connection speed we defined between the recorders and the on-prem servers.

Table 5.16: Throughput Maximum Input Constraints

Sub-System Connection	**Max (Gbps)**	**Mean (Gbps)**	**Std Dev (Gbps)**	**Cost**
Internet Package 1	1	0.8	0.1	$80/month
Internet Package 2	10	8	10	$150/month
Internet Package 3	100	80	100	$1,800/month
Recorder Output	100	90	10	-

Figure 5.95 shows the stochastic time series data we generated from Table 5.16 as input to the model for the simulations. We kept these time series inputs the same for every simulation to prevent differences in the OFVs due to different inputs. This is another advantage of a simulation that can take time series data as inputs.

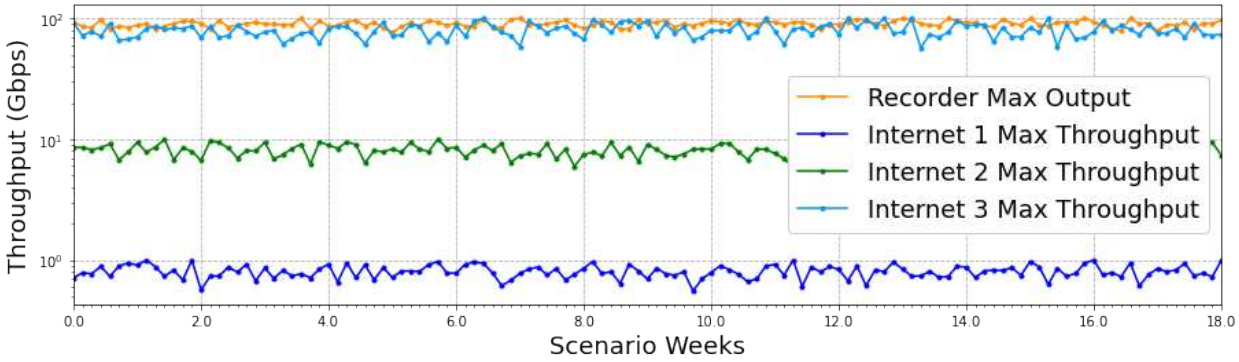


Figure 5.95: Randomized Recorder and Internet Package Maximum Throughput

In order to perform a sensitivity analysis later, we defined two different scenarios for this trade study. Scenario one represents an initially deployed hybrid cloud system that serves a single aircraft which records and uploads 10k GiB of data twice a week for engineers to analyze in the cloud environment. The second scenario represents two aircraft that fly five times a week with both uploading 10k GiB each flight. The time series inputs for both recorders are shown in Figures 5.96 and 5.97.

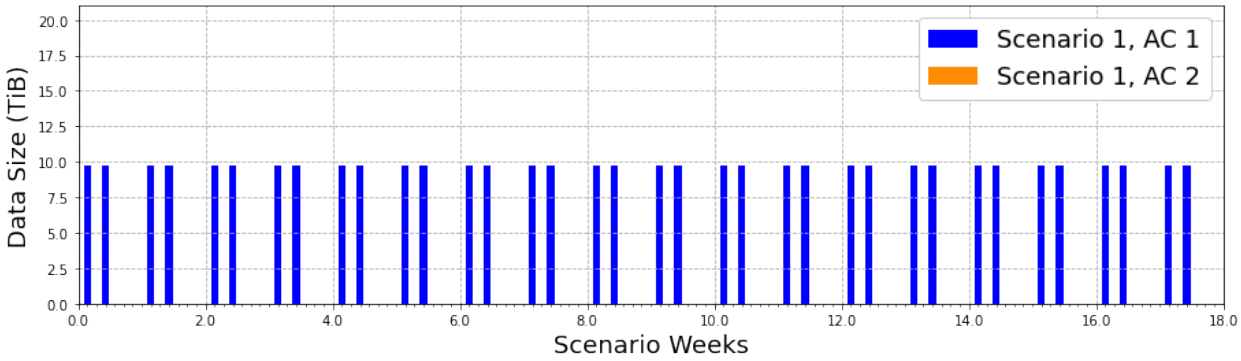


Figure 5.96: Scenario 1: One Aircraft Flying Twice a Week

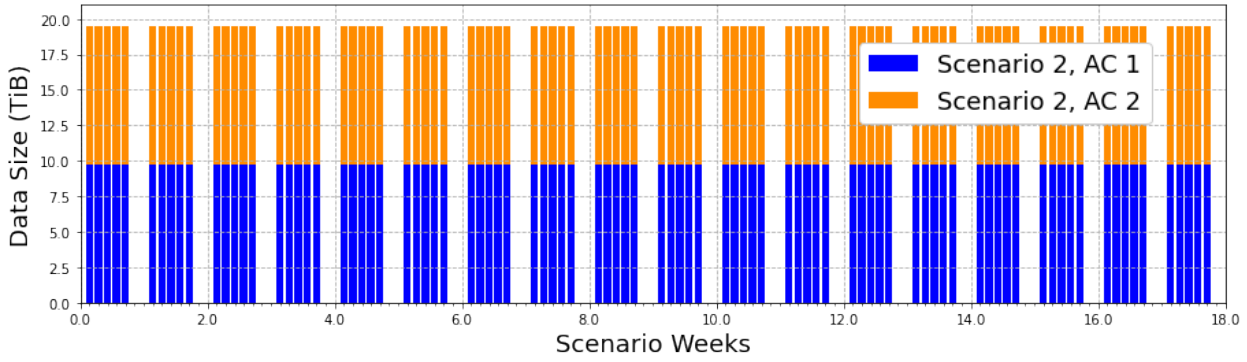


Figure 5.97: Scenario 2: Two Aircraft Flying Together Five Times a Week

The final trade study inputs that we defined were the initial purchases and recurring resource costs for the cloud and on-prem subsystems. We estimated the TCO using both CapExs and OpExs for on-prem local servers, local DI, commercial internet, and cloud services. These estimates are shown in Table 5.17.

The International Data Corporation (IDC) estimates TCO for data centers using an approximation of 20% hardware CapEx and the 80% left to hardware setup and OpEx [51]. We estimated the processing power, petabyte storage, and throughput requirements for the on-prem DI and found rough estimates online that would satisfy these requirements. We estimated the hardware costs of the CapEx to be $120k for compute and storage servers and another $48k for the rest of the DI. Using an estimated $83/hr for expert admin OpEx for 2 admins we calculated this would cost $28.8k/month ($345k/year). Using the IDC 20/80 approximation, we estimate the rest of the DI installation and server setup OpEx would cost $312k. Therefore, $480k CapEx and OpEx were estimated as initial purchases for the DI and local server simulation components. The rest of the monthly recurring charges are shown in Table 5.17.

Sections 5.7 and 5.7.3 present the results of the simulations, Pareto optimal solutions, and system analysis.

Table 5.17: Hybrid Cloud Initial Purchases & Recurring Costs (Model Ver.5)

Description	CapEx	OpEx
On-Prem DI:		
Local Server	$120k	-
System DI	$48k	-
Server & DI Setup	-	$312k
Admin Support	-	$28.8k/month
First Year Costs	$168k	$657k
Percentages	20%	80%
Internet:		
Internet Package 1	$500	$80/month
Internet Package 2	$500	$150/month
Internet Package 3	$1,000	$1,800/month
Cloud:		
Data Archive Service	-	$2,500/month
Cloud Enterprise Support	-	$33,000/month

5.7 Hybrid Cloud Case Study Results

This section describes the results and system analysis based on the two trade study scenarios we defined in Section 5.6.3. The results and our Pareto-Optimal design choice for scenarios 1 & 2 are shown in Sections 5.7.1 and 5.7.2, respectively. These two sections present plots of the Pareto and the fuzzy Pareto fronts for the three objective functions and also present tables of the Pareto-Optimal Solutions and their configurations. We also identify, through visual and quantitative methods, the dominant system characteristic and strong system characteristics, which helped identify our Pareto-Optimal design choice. The comparison of the two scenarios show sensitivity of our dominant feature and strong characteristics. Section 5.7.3 presents the results of the system analysis for the two scenarios. We used the Pareto-Optimal design choice compared against the prototype system performance and cost to calculate the potential cost savings over 1 year and 5 year periods.

5.7.1 Trade Study Scenario 1 Results

Table 5.18 contains the non-dominated Pareto front system simulations and the three min-max normalized OFVs for scenario 1. We've highlighted the Pareto-Optimal design choice and will discuss how we chose it later in this section. The results in this table shows that the three different `Max Pipe VM` values had no effect on scenario 1 results. This is shown in the simulation number column on the left side of the table which contains three simulations per row which have the same OFVs.

Table 5.18 results also show that data format 3 is a necessary condition for the Pareto front and indicating it is a dominant characteristic for scenario 1. Additionally, internet package 3 solutions are fully dominated and do not show up in this table. The results in this table also identifies that the solutions that never archive data do not show up on the Pareto front either. These results will be visualized throughout this section and includes a quantitative table of values that help identify dominate system characteristics.

Table 5.18: Scenario 1 Pareto Optimal Solutions OFVs (Model Ver.5)

Sim Num	Data Format	Max Pipe VMs	Internet Package	Days Until Archived	Archival Retention Days	Perf OFV	TVD OFV	Cloud Cost OFV
46,49,52	3	All	1	30	Forever	0.973169	0.781233	0.092441
47,50,53*	**3**	**All**	**2**	**30**	**Forever**	**0.9773**	**0.7804**	**0.0927**
73,76,79	3	All	1	60	Forever	0.9731	0.8598	0.0950
74,77,80	3	All	2	60	Forever	0.9773	0.8592	0.0953
100,103,106	3	All	1	30	30	0.9731	0.4593	0.0920
101,104,107	3	All	2	30	30	0.9773	0.4592	0.0922
154,157,160	3	All	1	30	60	0.9731	0.6312	0.0923
155,158,161	3	All	2	30	60	0.9773	0.6308	0.0926
181,184,187	3	All	1	60	60	0.9731	0.8413	0.0950
182,185,188	3	All	2	60	60	0.9773	0.8406	0.0953

**Pareto Optimal Design Choice*

Figure 5.98 shows the 3 dimensional Pareto-Optimal surface for our three objective functions calculated using the scenario 1 simulation results. A Pareto-Optimal curve (or surface) is one that

follows the minimum of the objective function on each axis and connects the points that minimizes one of the two (or three) objective function values while also minimizing or not increasing the other OFV. The fuzzy Pareto front consists of slightly dominated solutions that may outperform non-dominated solutions when other metrics or factors are considered. We analyzed both the non-dominated solutions and these slight dominated ones to find the Pareto optimal design choice and dominant system characteristics for this trade study.

Using red X symbols, Figure 5.98 highlights the non-dominated Pareto-Optimal solutions that define the Pareto front surface. The optimal design choice is also highlighted with a green circle. This design choice balances the maximum performance and TVD OFVs while also minimizing the cost OFV. All other points in this figure are the rest of the simulation results colored by cost OFV.

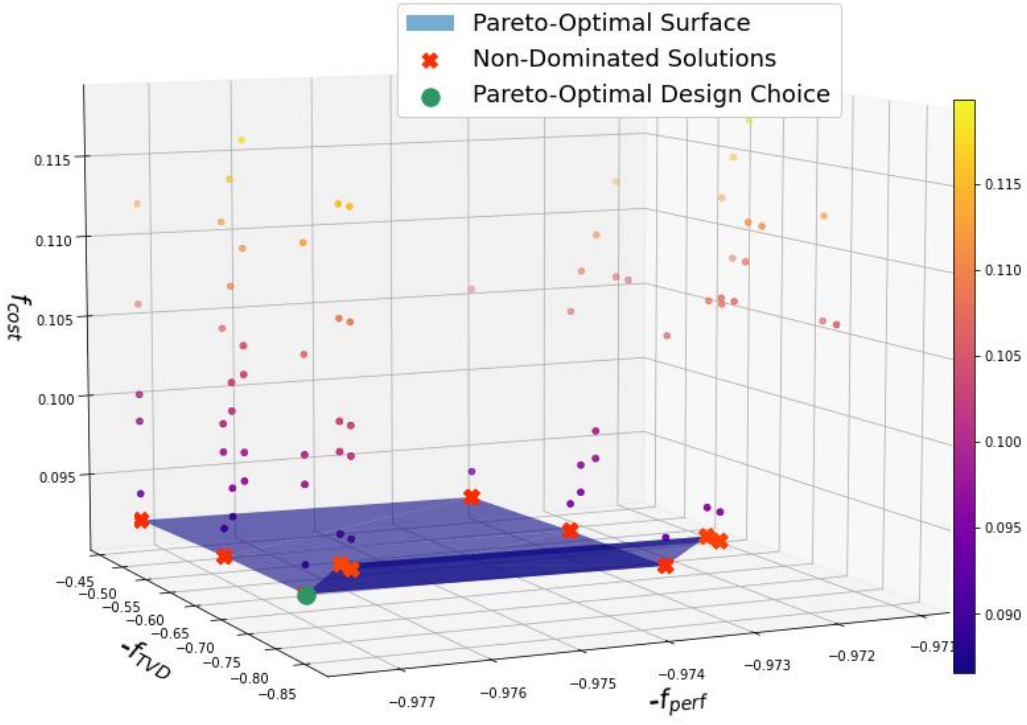


Figure 5.98: 3D Pareto-Optimal Solutions Surface View 1 (Scenario 1)

Figure 5.99 is the same data as Figure 5.98, but plotted at different angles to help visualize the 3 dimensional shape of the Pareto front surface. This viewing angle shows the steep increase in

price from the Pareto-Optimal design choice to the other 2 non-dominated solutions that increase the TVD OFVs. These last 2 non-dominated points in the upper left of the surface do not increase the performance at all. The design choice lies on Pareto front vertex which is the point on a Pareto front that most often represents the best value and balance between multiple objective functions. We chose this Pareto front vertex, or "knee-in-the-curve" as it is colloquially known, as our Pareto-Optimal design.

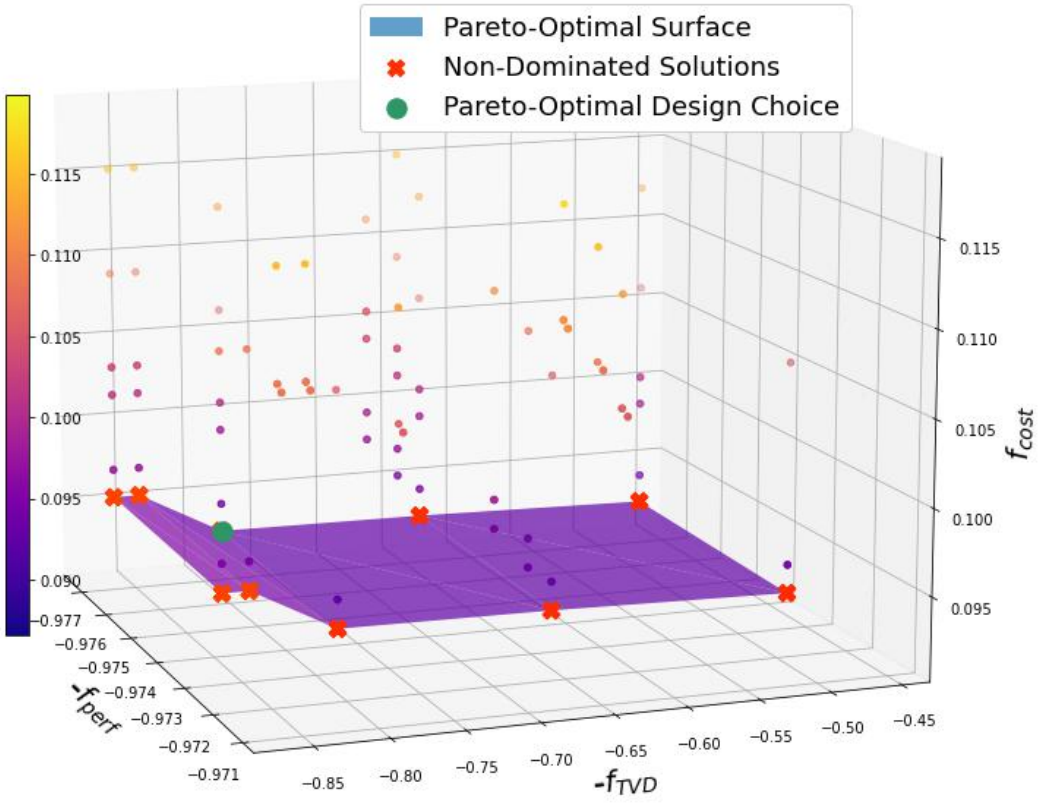


Figure 5.99: 3D Pareto-Optimal Solutions Surface View 2 (Scenario 1)

Figure 5.100 shows the scenario 1 solutions and Pareto front as a 2 dimensional plot of cost vs performance OFVs while also plotting the 3D Pareto surface solutions. We did this to highlight the differences in the Pareto front for 2 objective functions vs 3. This figure represents system if the stakeholder is only concerned about cost and performance. We plotted 2 levels of fuzzy Pareto frontiers to provide and example of the value of fuzzy Pareto solutions. We calculated each

fuzzy frontier level by removing the non-dominated Pareto-Optimal solutions and calculating the new non-dominated solutions. This figure shows that our 3D Pareto-Optimal design choice lies on the second level fuzzy Pareto front. We are showing this example to point out the importance of considering the fuzzy Pareto frontier in system optimizations. This example shows that when other objective functions or other aspects of the system are considered the optimal design choice may not lie directly on the Pareto front. In this example all Pareto and fuzzy Pareto solutions are part of the 3D Pareto surface, however not all 3D Pareteo surface solutions are on the 2D Pareto or fuzzy Pareto frontiers.

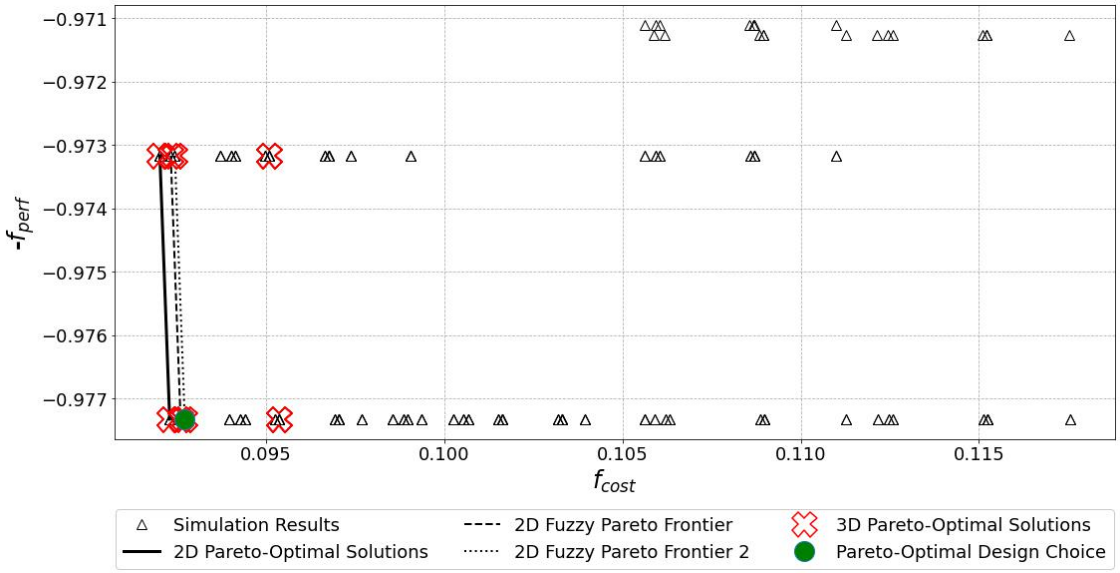


Figure 5.100: 2D Cost vs Performance vs 3D Pareto-Optimal Solutions (Scenario 1)

Table 5.19 shows one method of quantitatively identifying dominant or strong system characteristics. For example, the simulation matrix was comprised of 33% of data format 3 simulations, but the 3D Pareto-Optimal surface solutions for scenario 1 are 100% comprised of format 3. For scenario 1, using data format 3 is a requirement for being on the Pareto front, therefore this is an example of a dominant system characteristic.

The values we chose for the maximum pipe VMs decision variable are equally represented in the number of systems on the 3D Pareto surface compared to the number of systems with there

respective values. This indicates that the values chosen for this decision variable do not affect performance, cost, or TVD OFVs for this scenario.

Other decision variables have values that are over represented in the Pareto surface than they are in the simulation matrix. For instance, waiting 30 days until data is archived is represented in the Pareto surface more than the simulation matrix making this value for this feature a strong system characteristic, but not required for Pareto-Optimality. Waiting 60 days or never archiving data is under represented in the Pareto surface solutions. Another strong characteristic is retaining the archival data for only 60 days which is another over represented value in the Pareto surface. These strong system characteristics are the basis for forming data retention policies.

The internet package decision variable representation is an example where this table doesn't highlight a single strong characteristic, rather it highlights that internet package 3 never shows up on the Pareto surface for this scenario and therefore should not be considered for use. This conclusion is not always immediately apparent when considering a system that processes massive amounts of data. Many engineers we talked too assumed that we should install the internet package with the highest throughput to get the best performing system. This analysis shows that this assumption is incorrect for this scenario, however other scenarios may say otherwise. This is one of the main reasons we defined the second scenario with nearly an OOM more data to process in a week. This second scenario is discussed in Section 5.7.2.

In Figures 5.102 to 5.106 we visualize examples of clusters in the trade space which provide additional context to the emergent properties of each decision variable and their values we chose for this scenario.

Figure 5.101 shows the 2D cost and performance objectives with the data formats colored differently to show the trade space clusters. The trade space cluster of data format 2 shows a distinctly higher cost and lower performance than the data formats 1 & 3 which are intermingled. Data format 3 lies on the 2D Pareto and fuzzy Pareto frontiers and is a requirement of being on the Pareto surface. This figure shows the relative differences between data format 3 and data format 1 in cost. These two formats appear to have the same performance for the wide variety of other

Table 5.19: Scenario 1: Percentages of Systems Along Pareto Front (Model Ver.5)

Decision Variable	Value	% of Systems w/Feature	% of 3D Pareto-Optimal Systems w/Feature
Data Format	1	33.3%	0%
	2	33.3%	0%
	3	33.3%	100%
Max Pipe VMs	5	33.3%	33.3%
	10	33.3%	33.3%
	20	33.3%	33.3%
Internet Package	1	33.3%	50%
	2	33.3%	50%
	3	33.3%	0%
Days Until Archived	30	42.9%	60%
	60	42.9%	40%
	Never	14.3%	0%
Archival Retention Days	30	28.6%	20%
	60	28.6%	40%
	Forever	42.9%	40%

decision variables. Figure 5.102 shows the same simulation data sorted by data format, but plotted on the TVD and performance objective function axes. The TVD OFVs are the same among all three data formats however the format 2 still has outliers in slow performance at the top of the plot.

Figures 5.103 and 5.104 are colored by internet package to show the trade space clusters. Figure 5.103 shows that internet package 1 cluster has less performance than the other two. The least performant simulations in the upper right corner are associated with data format 2 as previously seen in Figure 5.101. Figure 5.103 shows internet packages 2 & 3 have the same performance, but are separated by cost. The separation by performance is also seen in Figure 5.104, however the TVD OFV does not appear to be affected by internet package.

Figures 5.105 and 5.106 both show a comparison of the simulation results for cost and TVD functions. Figure 5.105 is colored by the number of days until data is archived. This figure shows that waiting 30 days until archiving data dominates the Pareto and fuzzy Pareto front which makes it a strong system characteristic for data retention policies for this scenario. It also shows the 30

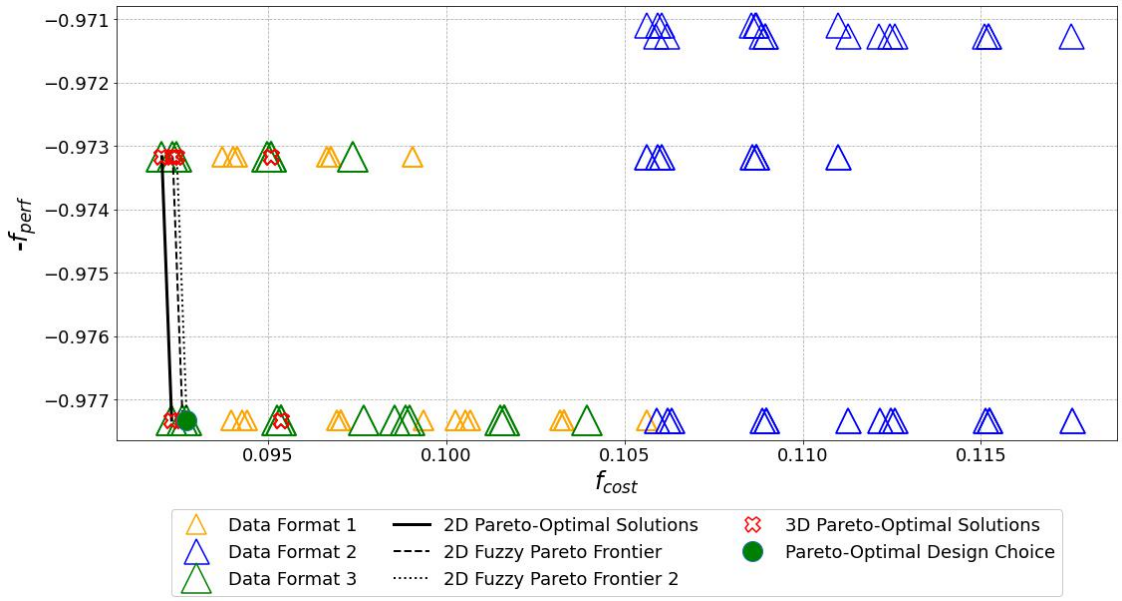


Figure 5.101: Cost vs Performance Trade Space Clusters by Data Format (Scenario 1)

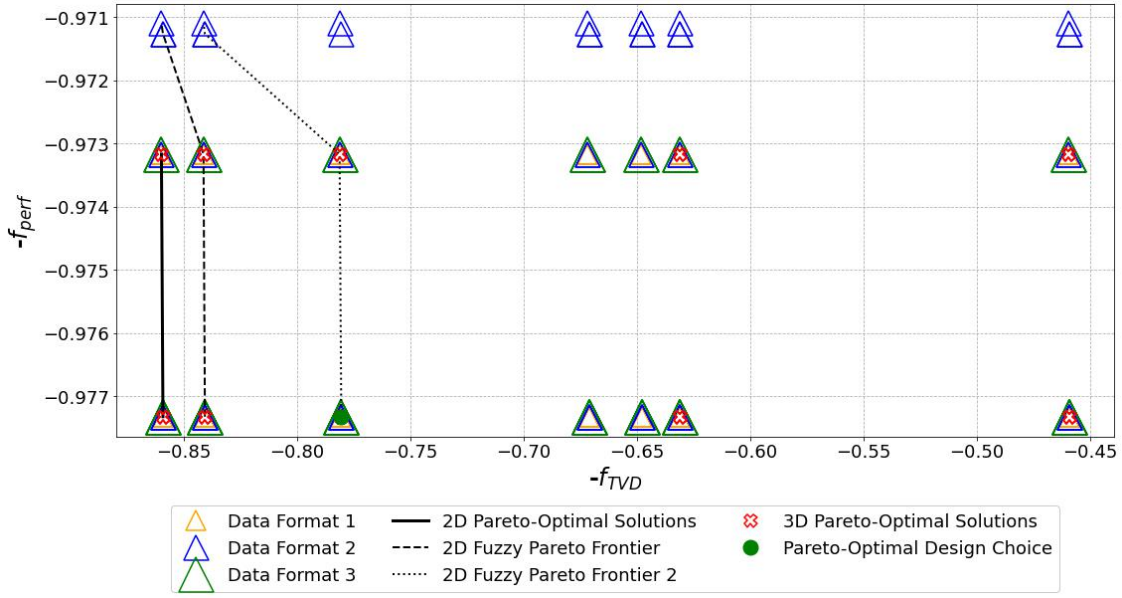


Figure 5.102: Performance vs TVD Trade Space Clusters by Data Format (Scenario 1)

day setting is the cheapest trade space. However, waiting 60 days to archive data maximizes the TVD trade space if that is weighted more than other objectives.

Figure 5.106 shows the results colored by days for archival data retention. These results show the indefinite retention of archival data dominates the maximimum TVD value trade space when plotted on these axes. However, Table 5.19 results show that the strong system characteristic is

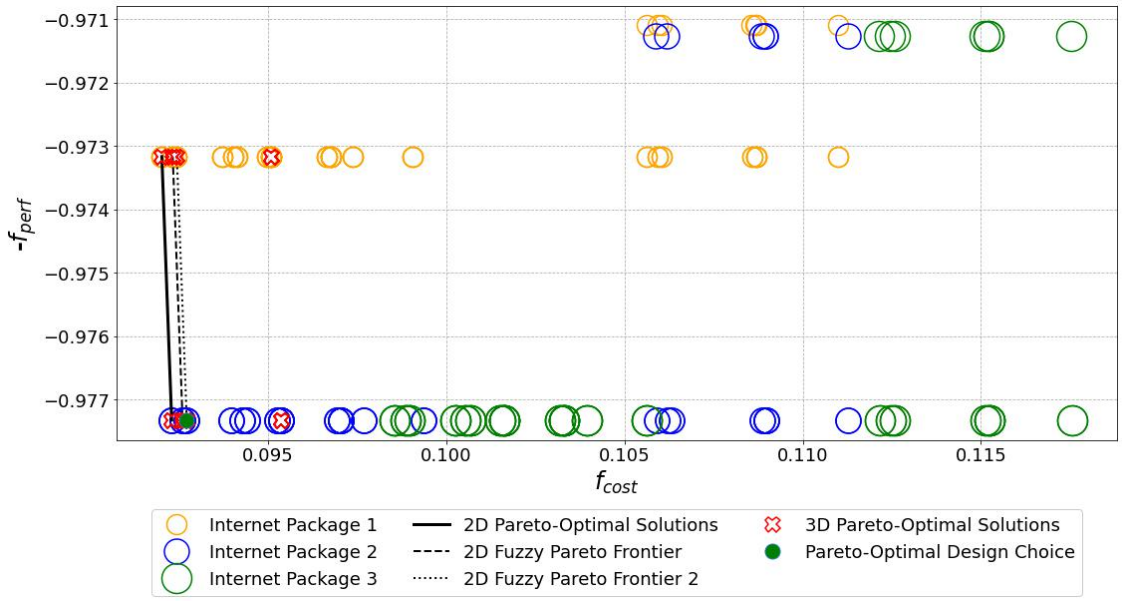


Figure 5.103: Cost vs Performance Trade Space Clusters by Internet Package (Scenario 1)

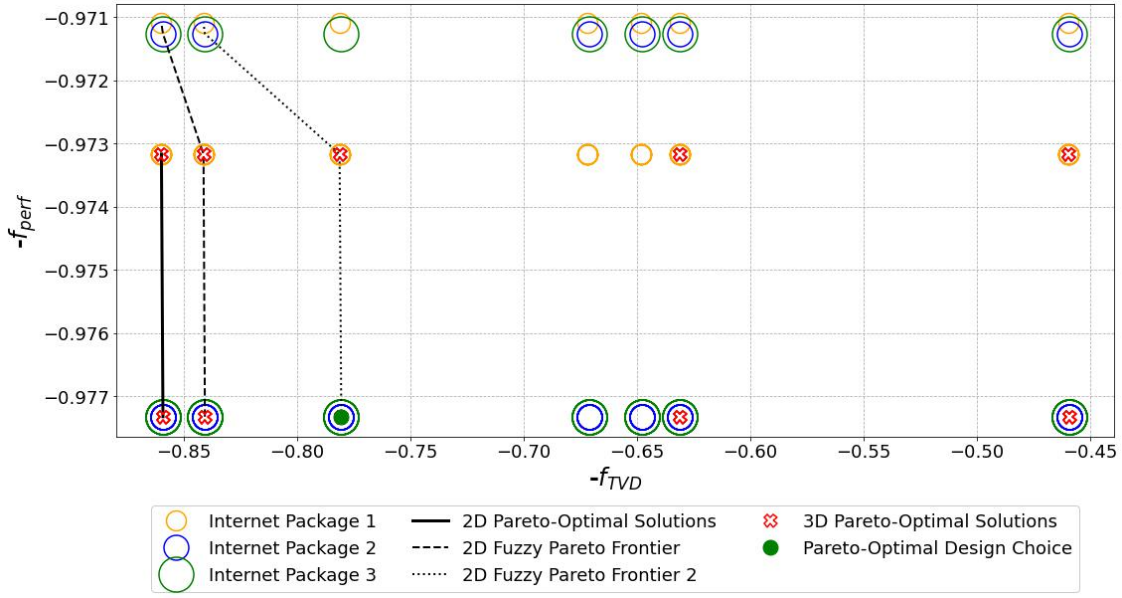


Figure 5.104: Performance vs TVD Trade Space Clusters by Internet Package (Scenario 1)

to archive for 60 days. This is a good example of using quantitative and visual methods when analyzing Pareto results.

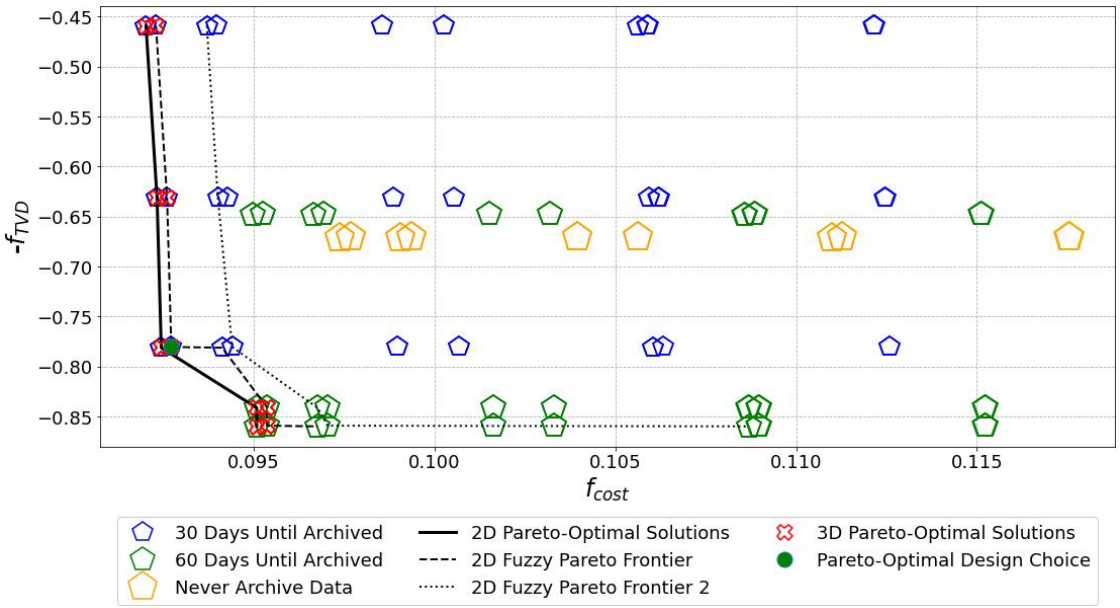


Figure 5.105: Cost vs TVD Trade Space Clusters by Days Until Archived (Scenario 1)

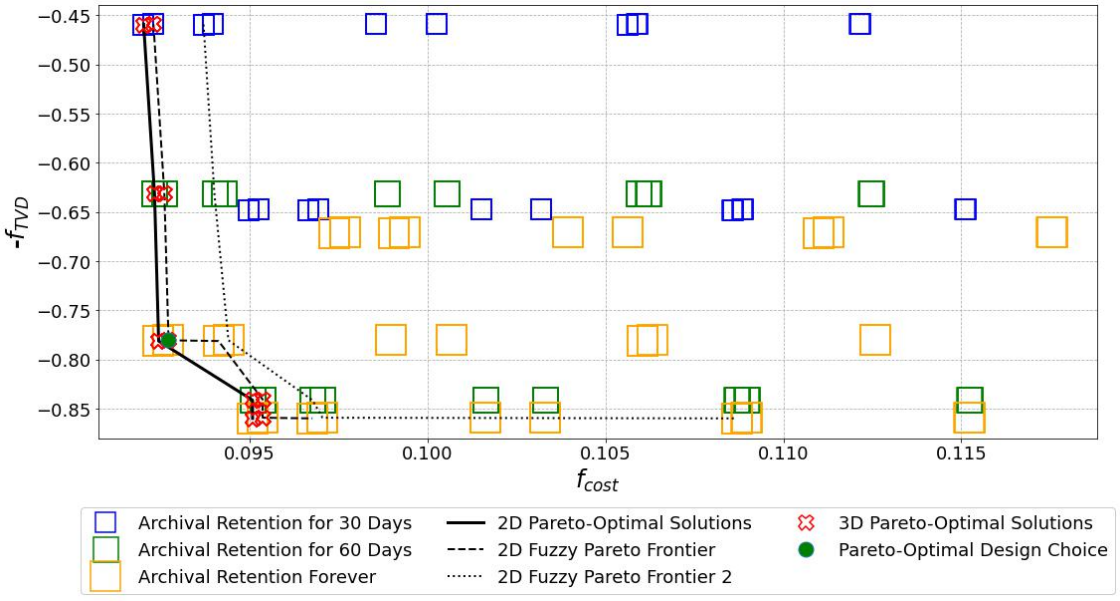


Figure 5.106: Cost vs TVD Trade Space Clusters by Days for Archival Retention (Scenario 1)

5.7.2 Trade Study Scenario 2 Results

Table 5.20 contains the non-dominated Pareto surface system simulations and the three min-max normalized OFVs for scenario 2. Scenario 2 increased the upstream data input size from 1 aircraft twice a week to 2 aircraft five times a week compared to scenario 1. We did this to analyze the sensitivity of the Pareto front and dominant system characteristics for changes in aircraft data

recording frequency. The slight differences in this table from scenario 1 results indicate the sensitivities to an increase in recorded data input. These changes are largely due to the inclusion of data format 2 solutions on the Pareto surface. The reason for the addition of this least performant and most costly format to the Pareto surface is explained later in this section, but it was largely due to their maximized TVD OFVs. We've also highlighted the Pareto-Optimal design choice in this table which was chosen using the Pareto surface vertex nearest the minimized OFVs in the same manner as we did for scenario 1. This design choice is the same as scenario 1 which indicates it is not sensitive to an increase in recorded data input into the system.

Table 5.20: Scenario 2 Pareto Optimal Solutions OFVs (Model Ver.5)

Sim Num	Data Format	Max Pipe VMs	Internet Package	Days Until Archived	Archival Retention Days	Perf OFV	TVD OFV	Cloud Cost OFV
38,41	2	5,10	2	30	Forever	0.2696	0.7850	0.1767
46,49,52	3	All	1	30	Forever	0.3453	0.4380	0.1356
47,50,53*	**3**	**All**	**2**	**30**	**Forever**	**0.9547**	**0.7850**	**0.1890**
65,68	2	5,10	2	60	Forever	0.2696	0.8628	0.1898
73,76,79	3	All	1	60	Forever	0.3453	0.4822	0.1429
74,77,80	3	All	2	60	Forever	0.9547	0.8628	0.2021
100,103,106	3	All	1	30	30	0.3453	0.2548	0.1345
101,104,107	3	All	2	30	30	0.9547	0.4602	0.1870
146,149	2	5,10	2	30	60	0.2696	0.6406	0.1761
154,157,160	3	All	1	30	60	0.3453	0.3598	0.1353
155,158,161	3	All	2	30	60	0.9547	0.6406	0.1885
173,176	2	5,10	2	60	60	0.2696	0.8479	0.1898
181,184,187	3	All	1	60	60	0.3453	0.4760	0.1429
182,185,188	3	All	2	60	60	0.9547	0.8479	0.2021

**Pareto Optimal Design Choice*

Table 5.21 includes the scenario 2 quantitative measure of the strength of the decision variable settings. This table shows minor differences in dominance compared to scenario 1 results in Table 5.19. This indicates some sensitivity to data input rate for the dominant features. For instance, data format 3 is no longer a necessary condition for the Pareto surface, instead, data format 2 solutions are now present on the Pareto surface.

Table 5.21: Scenario 2: Percentages of Systems Along Pareto Front (Model Ver.5)

Decision Variable	Value	% of Systems w/Value	% of 3D Pareto-Optimal Systems w/Feature
Data Format	1	33.3%	0.0%
	2	33.3%	11.8%
	3	33.3%	88.2%
Max Pipe VMs	5	33.3%	41.2%
	10	33.3%	29.4%
	20	33.3%	29.4%
Internet Package	1	33.3%	44.1%
	2	33.3%	55.9%
	3	33.3%	0.0%
Days Until Archived	30	42.9%	58.8%
	60	42.9%	41.2%
	Never	14.3%	0.0%
Archival Retention Days	30	28.6%	17.6%
	60	28.6%	41.2%
	Forever	42.9%	41.2%

Additionally, the number of maximum pipeline VMs now affects which solutions are optimal compared to scenario 1 which had no affect. The table shows that 5 maximum number of pipes is now a strong characteristic, however we'll show later that this is a direct result of the maximized TVD OFVs from data format 2 and not necessarily a characteristic that should be chosen. This is a good example that this quantitative analysis does not provide the full solution and other methods such as visual plots should be used as well.

Table 5.21 results show that internet package 2 is now a stronger characteristic than package 1 as a result of the increase in data ingest into the system. The plots later in this section show that internet package 2 is also dominant on the fuzzy Pareto frontier. This table also shows that the data archive and retention settings are less sensitive to the increase in data system ingest size than the other decision variables. The strong characteristics have remained the same compared to scenario 1.

Figures 5.107, 5.108, and 5.109 show the 3D Pareto surface for scenario 2. This Pareto surface includes more contours and is more complex to view than the surface for scenario 1. For this

reason, we chose to include 3 different views of the surface. In these figures we show the data format 2 points along the Pareto front which shows that they have the worst performance of the Pareto surface solutions.

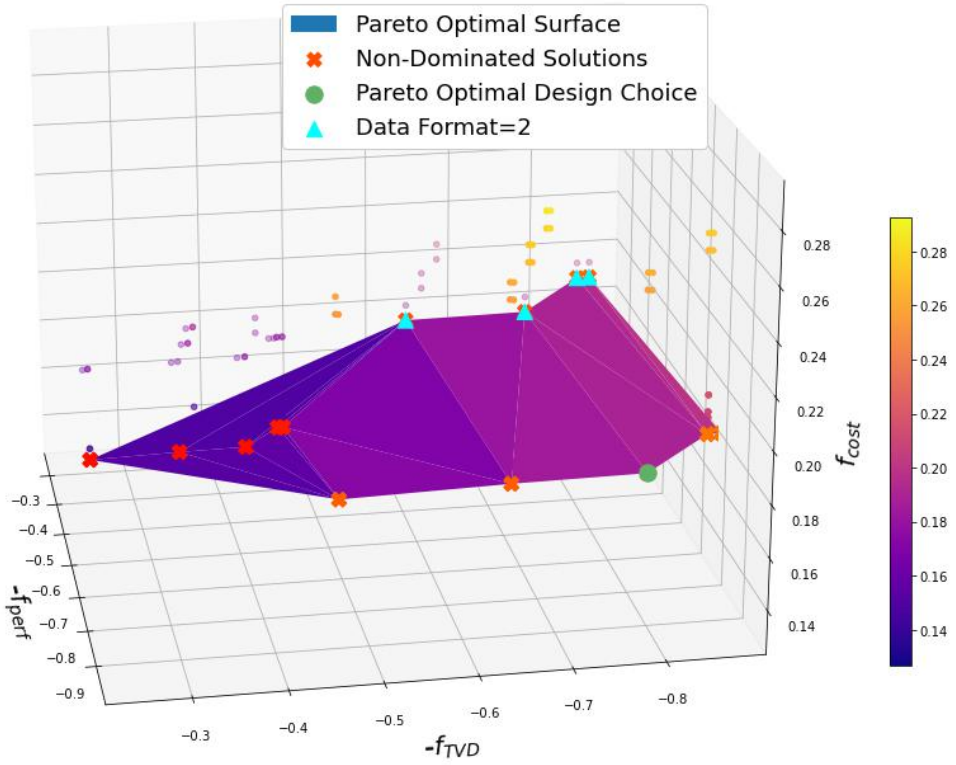


Figure 5.107: 3D Pareto-Optimal Solutions Surface View 1 (Scenario 2)

Figure 5.108 shows a view that emphases the scenario 2 larger range in the f_{cost} axis. This view also helps visualize the Pareto surface convex hull. Figure 5.109 provides a viewing angle that emphasizes performance and cost of the optimal design choice. This design choice, which is the same as scenario 1, is on the leading edge of the best performing solutions. This is also located at the vertex of this surface's edge where the costs increase without an increase to performance. This point is again located at and ideal balance between cost, performance, and TVD.

Figure 5.110 shows a similar trend in trade space clusters for scenario 2 compared to scenario 1 for data formats when looking at cost and performance only. The Pareto and fuzzy Pareto frontiers contain only data format 3. For scenario 2, data format 2 continues to cost the most and perform the

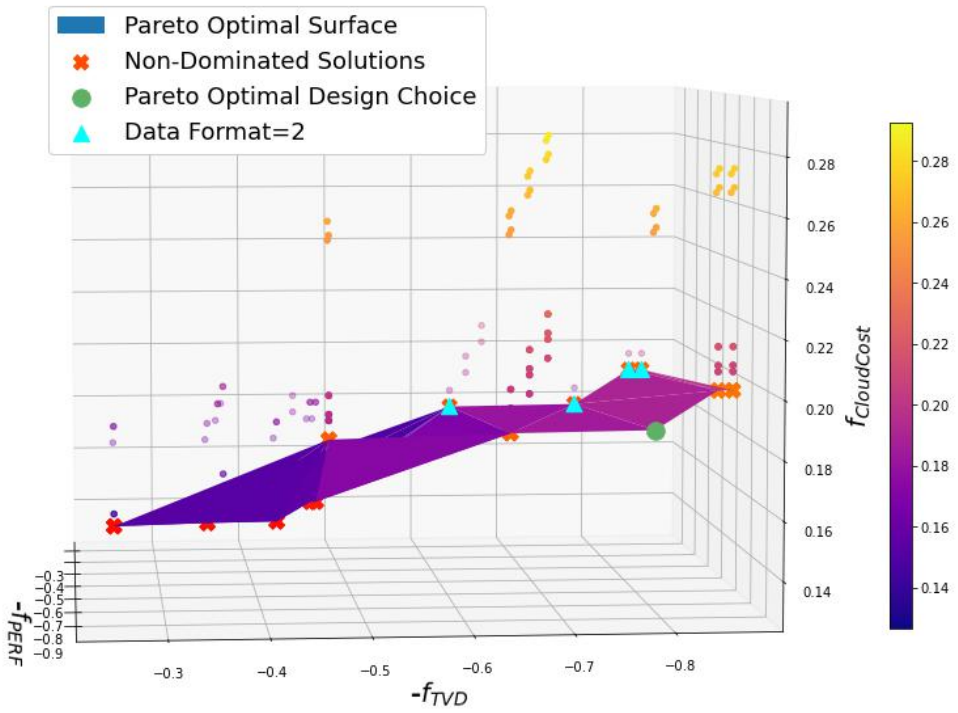


Figure 5.108: 3D Pareto-Optimal Solutions Surface View 2 (Scenario 2)

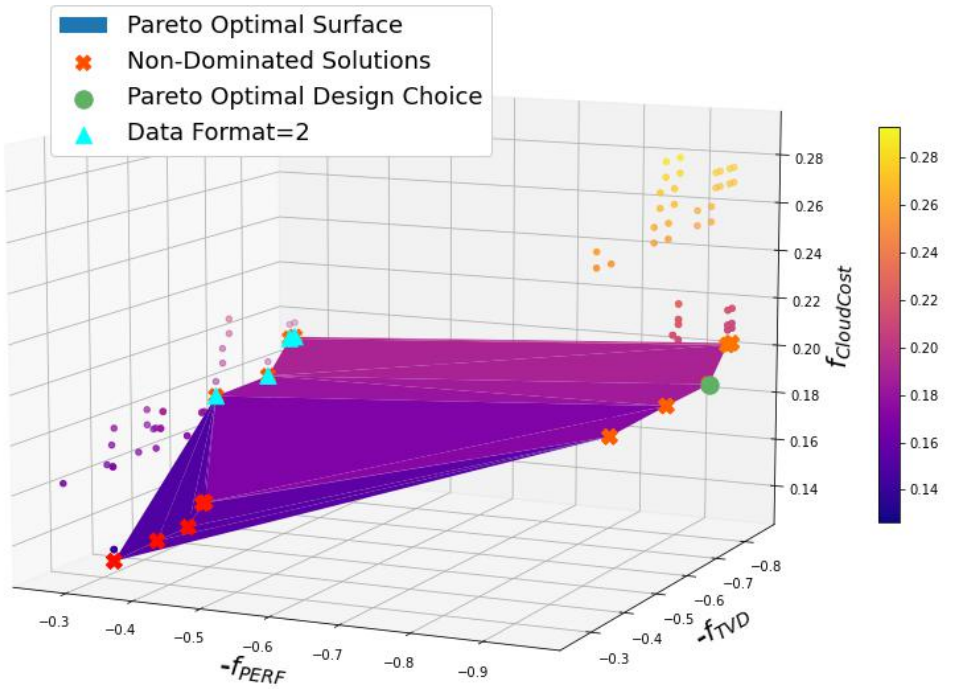


Figure 5.109: 3D Pareto-Optimal Solutions Surface View 3 (Scenario 2)

worst, however the 3D Pareto surface solutions are shown within the upper most trade space cluster. Figure 5.111 shows that these data format 2 solutions are on the fuzzy Pareto front when looking at TVD and performance. When ignoring costs, some data format 2 solutions are intermingled with the other data formats.

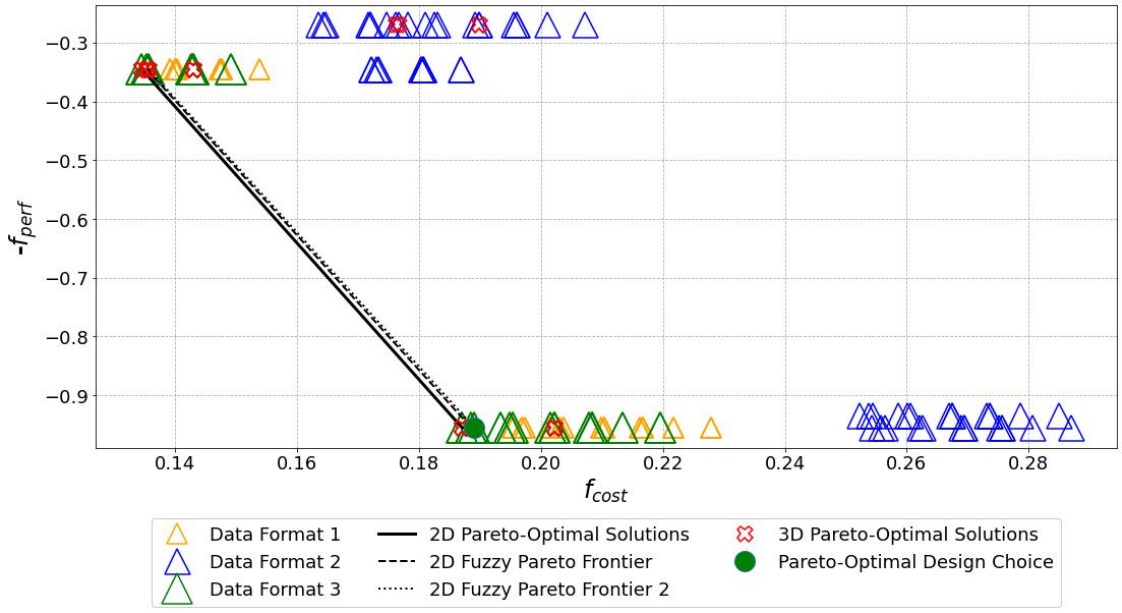


Figure 5.110: Cost vs Performance Trade Space Clusters by Data Format (Scenario 2)

Figure 5.112 shows the scenario 2 Pareto and fuzzy Pareto frontiers for cost and TVD for the 3 different data formats. The data format 2 trade space clusters are still some of the most expensive, however there are some solutions along the Pareto and fuzzy Pareto frontier. This figure is another example that illustrates the importance of considering other objective functions that can result in the optimal design choice coming from the fuzzy Pareto frontier.

Figure 5.113 shows the expanding difference in performance between internet package 1 and the other two. This is more prevalent for the increase in data ingestion for scenario 2 than it was for scenario 1. Internet package 2 is now a strong system characteristic and continues to perform the same as package 3. Figure 5.114 shows the trade space cluster of internet package 1 in the upper right corner of the TVD and performance axes, which indicates this is a characteristic to avoid unless cost is weighted stronger than any other objective.

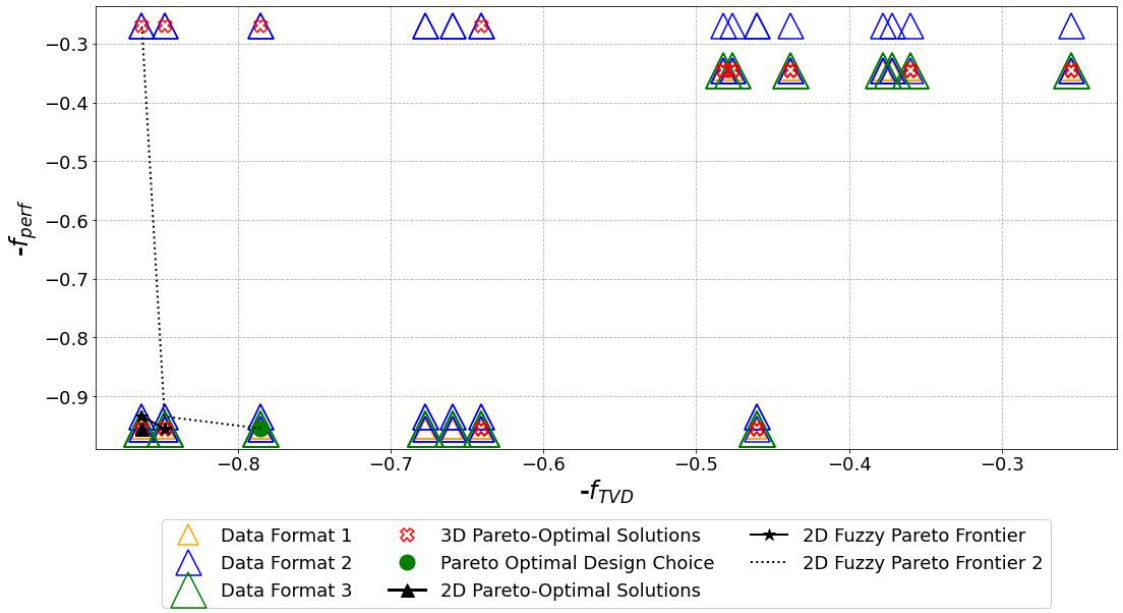


Figure 5.111: Performance vs TVD Trade Space Clusters by Data Format (Scenario 2)

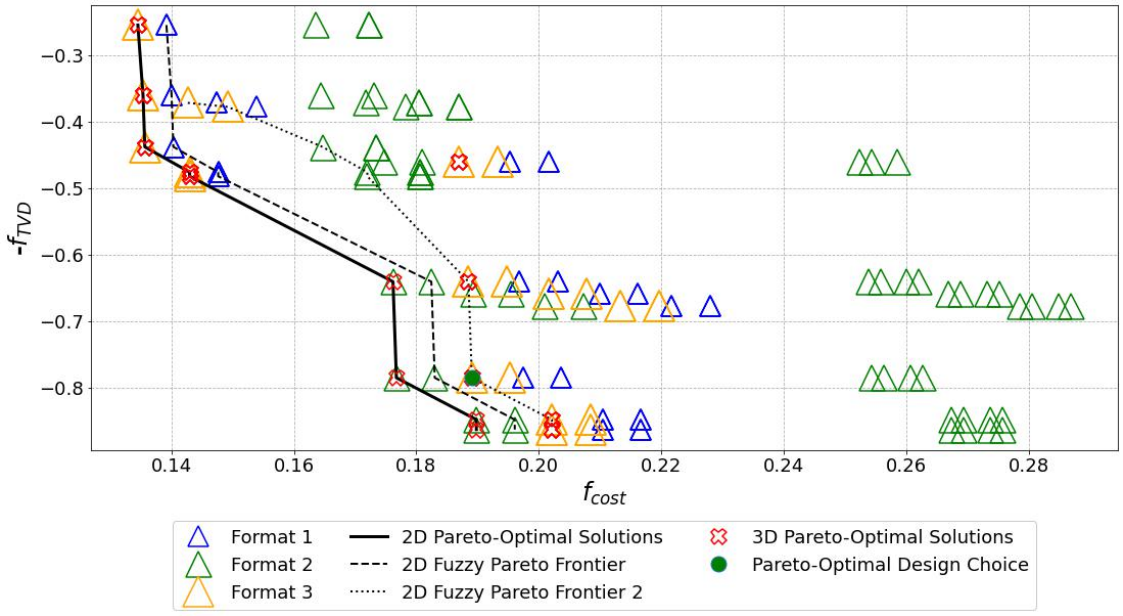


Figure 5.112: Cost vs TVD Trade Space Clusters by Data Format (Scenario 2)

Figures 5.115 and 5.116 show the mixture of trade space clusters colored by the data archive days. These figures show that 30 days to archive data continues to be the strongest system characteristic when comparing cost, performance, and TVD. Figure 5.115 shows that archiving data has very little affect on performance. On the other hand, Figure 5.116 shows that this strong system

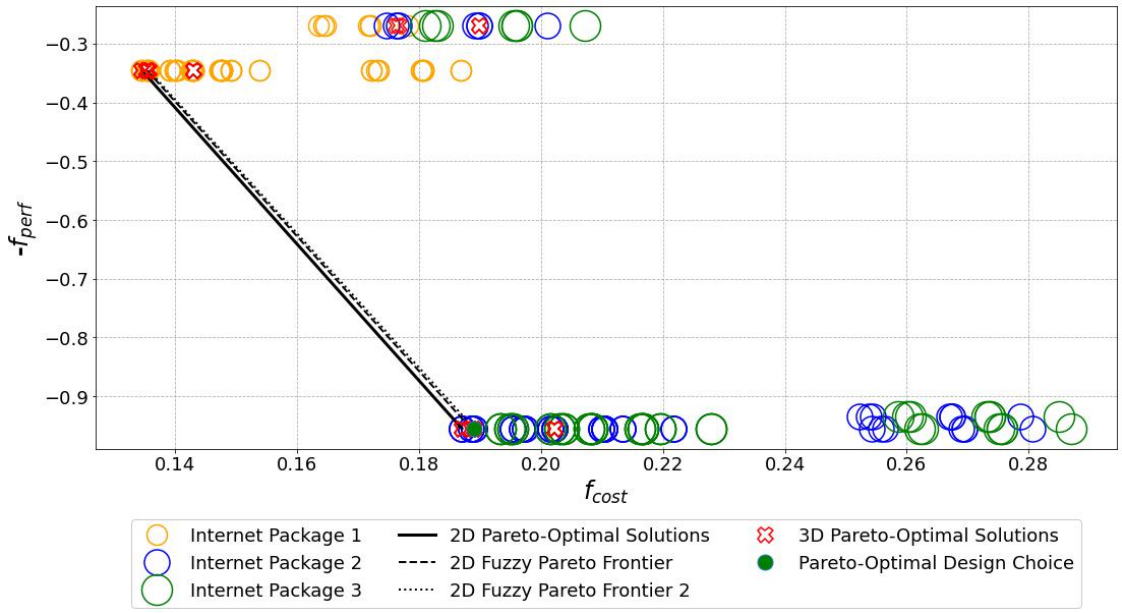


Figure 5.113: Cost vs Performance Trade Space Clusters by Internet Package (Scenario 2)

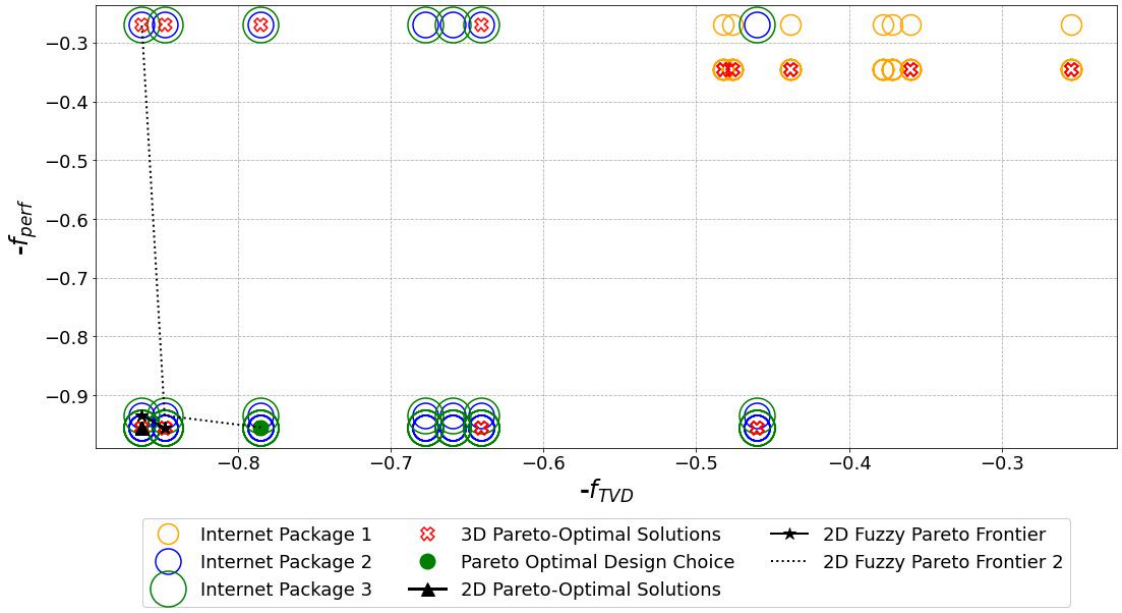


Figure 5.114: Performance vs TVD Trade Space Clusters by Internet Package (Scenario 2)

characteristic is easier to see when plotting TVD vs cost. This figure shows that 30 days to archive dominates the Pareto and fuzzy Pareto frontiers.

Figure 5.117 shows the trade space clusters colored by the archival retention days. 60 days of archival retention is the strongest characteristic/setting for values as listed in Table 5.21 due

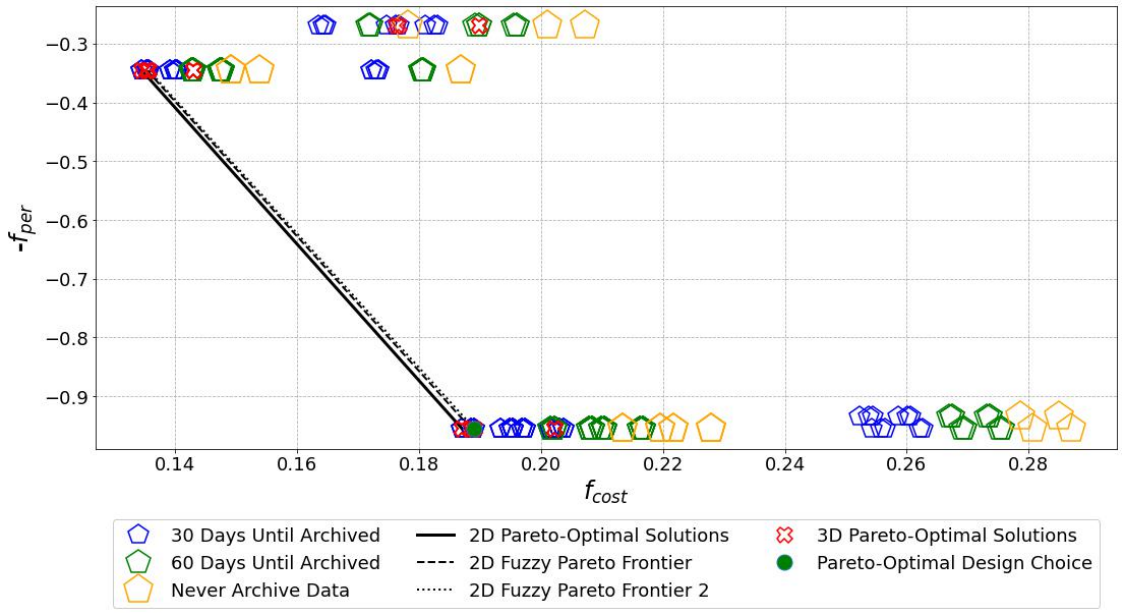


Figure 5.115: Cost vs Performance Trade Space Clusters by Days Until Archived (Scenario 2)

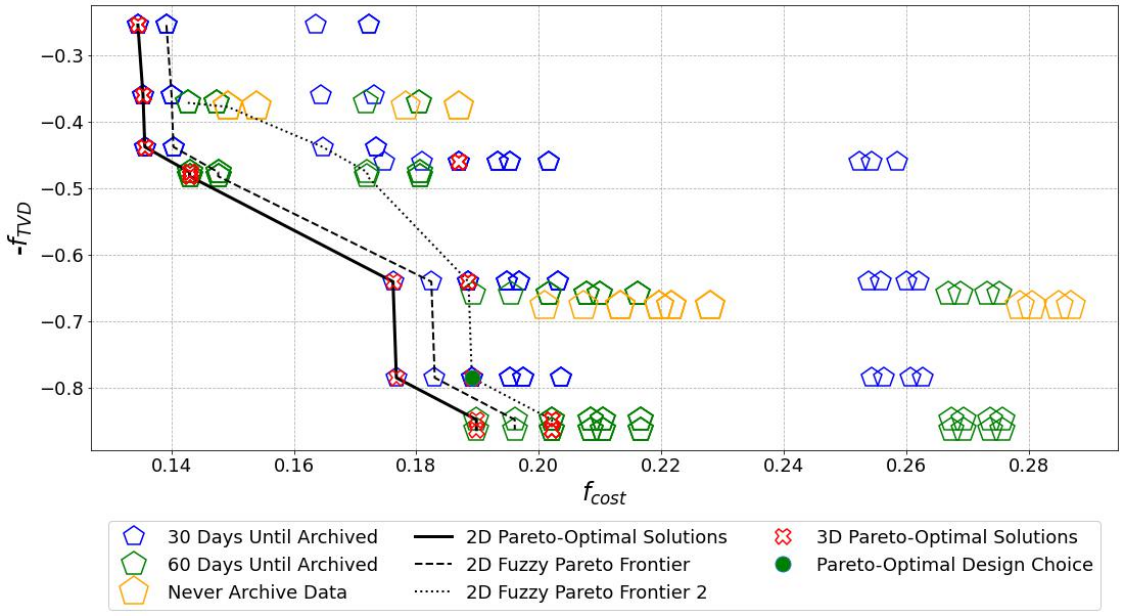


Figure 5.116: Cost vs TVD Trade Space Clusters by Days Until Archived (Scenario 2)

to the increase in system percentage to Pareto front percentage. However, retaining archive data "forever" over the 18 week simulation had little affect on the system cost and therefore this setting was chosen for the optimal design choice. Ideally, a longer simulation would show a larger change in cost for indefinite archive data retention, so this is an area to refine in the future. This is a good

example of analyzing your assumptions before deploying your optimal solution to a large scale system.

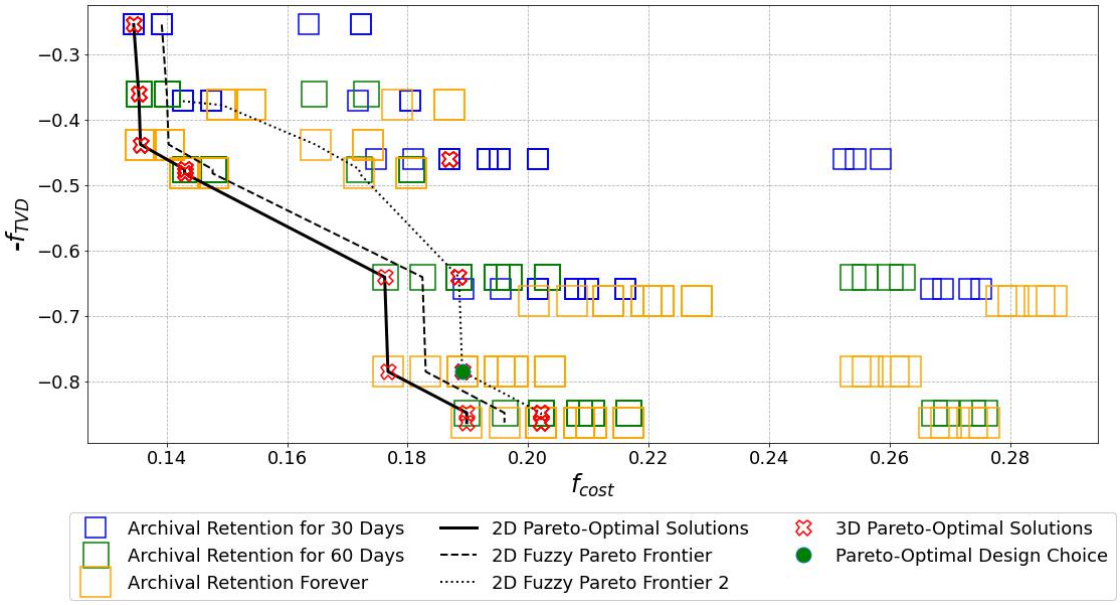


Figure 5.117: Cost vs TVD Trade Space Clusters by Days for Archival Retention (Scenario 2)

5.7.3 System Analysis Results

In this section we summarize the trade study scenario results and compute the cost savings of the optimal design choice compared to the prototype hybrid cloud costs for 1 and 5 year periods. The dominant system characteristic prevalent in both scenarios was data format 3. Each format analyzed was a combination of different recording formats, communication protocols, ETL pipeline software, and ETL pipeline storage strategies. The strong system characteristics for both scenarios were internet package 2 and archiving data after 30 days. The optimal archival data retention settings are not clear since the quantitatively strong characteristic was to delete this data after 60 days, however the optimal design choice never deleted this data. Therefore, additional simulation should be focused around extended simulations and more archival retention choices. The analysis of the Pareto results provided the ability to further focus on these dominant and strong characteristics which reduces the trade space to a more manageable size.

The rest of this section will be focused on forecasting long term savings of the optimal designs compared to the prototype design for the two previously defined scenarios. Since most organizations care about their annual budget and their longer term costs we analyze these savings for 1 and 5 year time frames. Additionally, we compare the 3D Pareto-optimal design choice against the Pareto-optimal design if cost and performance were the only objectives. This comparison provides a 1 and 5 year cost estimate of the TVD. The Pareto-optimal cost/perf design is identical to the 3D design except it deletes archival data after 30 days.

Since our current UM simulation only ran for 18 weeks we had to find an algorithm that could accurately forecast these results to 1 and 5 years for both scenarios. Figures 5.118 and 5.119 show two methods we used for each scenario, respectively, to forecast the simulations out for 1 year. The initial forecast algorithm used a 2nd-order polynomial model. The second forecast algorithm was comprised of several simple algorithms based on different simulation settings, therefore we called this our forecast meta-algorithm. This meta-algorithm used the decomposed component results of the simulation to produce an accurate forecast. For example, it uses archive and retention timelines, recurring costs, recurring cost timelines, and linear fits to daily time series component costs as inputs to produce this accurate solution.

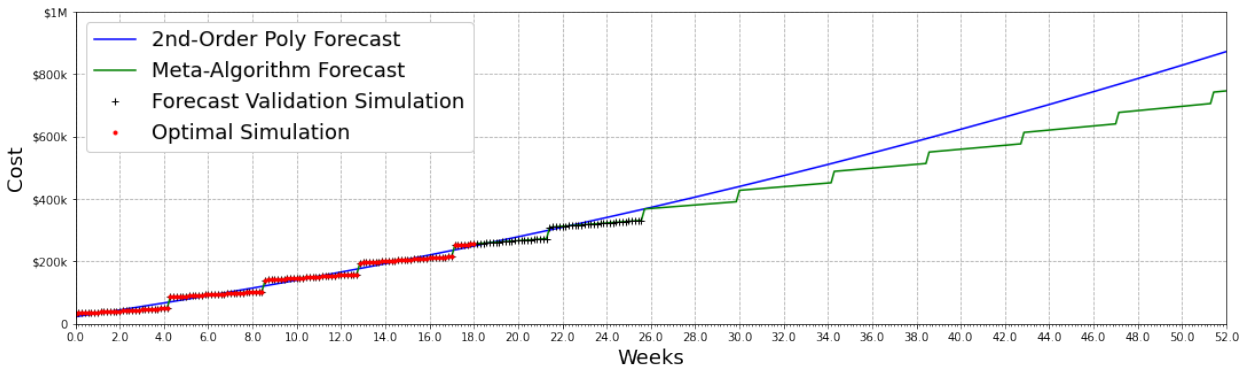


Figure 5.118: Forecast Algorithms for Simulation Extension (Scenario 1)

We ran 2 new simulations for 180 days to produce a validation data set in which to quantify the accuracies of our 2 forecasting algorithms. This validation data set is necessary since we use

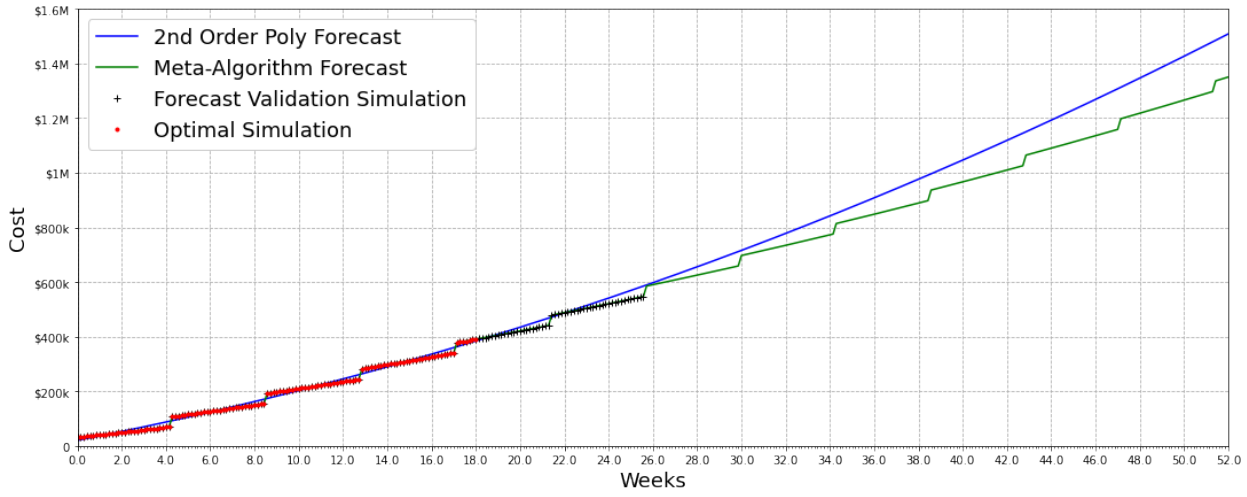


Figure 5.119: Forecast Algorithms for Simulation Extension (Scenario 2)

the previous 127 day simulations to train our models. We chose to run the 3D optimal design for both scenarios. Figures 5.118 and 5.119 show the two algorithms, the 127 day optimal simulation, and the 180 day validation data set. These visual results show that the meta-algorithm forecast method matches both optimal simulation and validation data sets. Table 5.22 shows the quantitative measurements of these two forecasting algorithms against the validation data set for both scenarios. These results show that our forecasting meta-algorithm is highly accurate with an RMSE three OOMs lower than the total cost.

Table 5.22: Simulation Forecasting Algorithm Validation Accuracies

Forecast Technique	Total Cost Percent Error	RMSE	Entropy	Validation Total	Forecast Total
Scenario 1					
Meta-Algorithm	0.01%	$246	3.4	$331,750	$331,707
2nd-Order Poly	10.3%	$12,311	4.1	$331,750	$366,005
Scenario 2					
Meta-Algorithm	0.05%	$1,029	2.9	$546,857	$547,157
2nd-Order Poly	6.99%	$13,039	4.0	$546,857	$585,075
Max Entropy: 4.6					

The Anderson-Darling (AD) and D'Agostino-Pearson (DP) [140] tests for normality of the meta-algorithm forecast residuals have values of 0.026 and 0.159, respectively, which indicates that these residual are normally distributed. Therefore, it is a valid assumption that we use the RMSE values from Table 5.22 to for our Root Mean Squared Forecasting Error (RMSFE). Multiplying this RMSFE by 1.96 gives us our 95% prediction interval. Figures 5.120 and 5.121 show a close-up view of our meta-algorithm forecast solutions, 95% prediction interval, optimal simulation, and the validation data set. These results show that the validation data set lies within our 95% prediction interval for both scenarios.

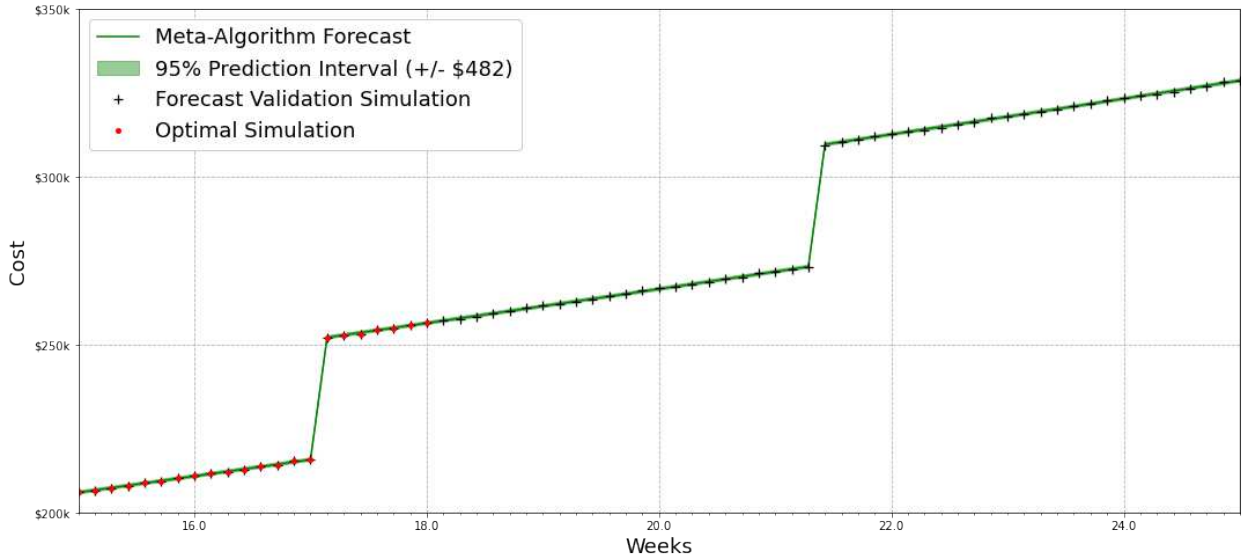


Figure 5.120: Forecast Validation with Prediction Interval for 3D Pareto-Optimal Design (Scenario 1)

Figures 5.122 and 5.123 show the comparisons of the prototype configuration, 3D Pareto-Optimal design, and the cost and performance Pareto-optimal design annual costs for both scenarios. These figures indicate a significant annual cost savings for both Pareto-optimal designs compared to the prototype configuration. They also show a significant increase in annual cost savings between scenario 1 and 2. This indicates the cost savings are sensitive to the amount of recorded data processed through the data system as we expected.

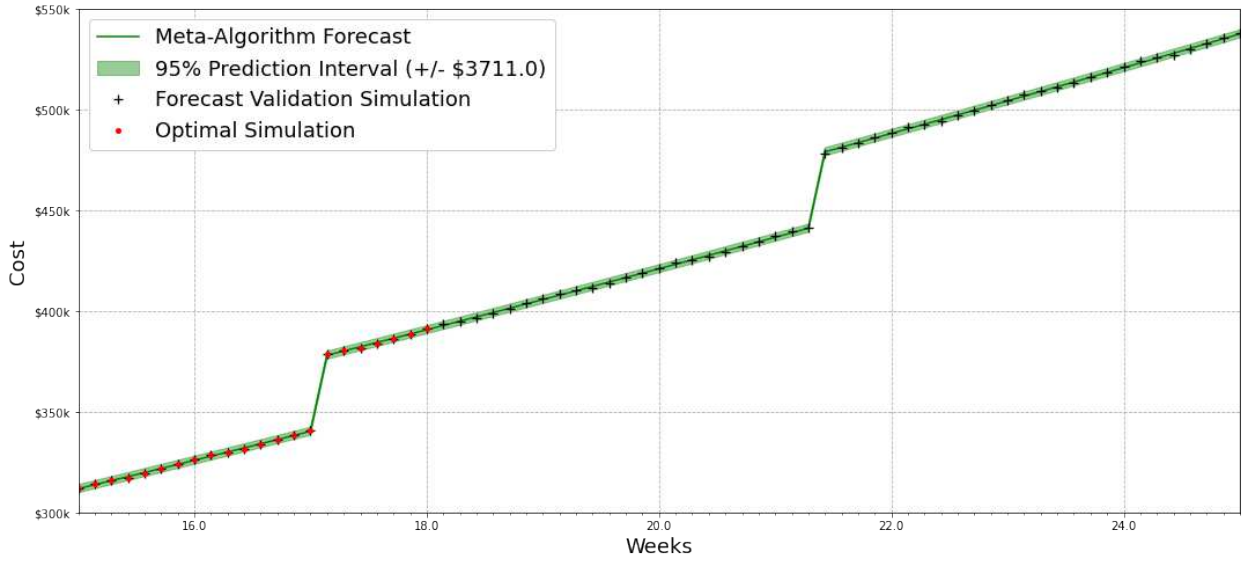


Figure 5.121: Forecast Validation with Prediction Interval for 3D Pareto-Optimal Design (Scenario 2)

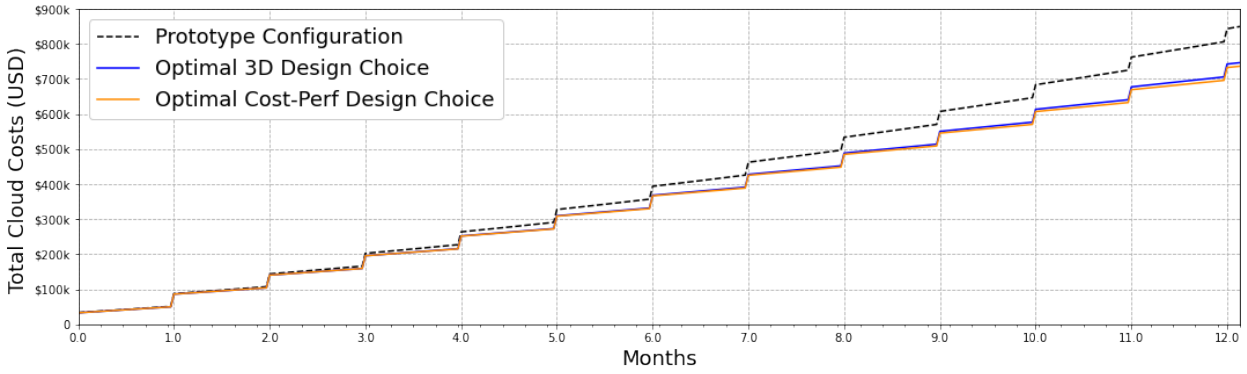


Figure 5.122: Prototype vs 3D and Cost/Perf Pareto-Optimal Solutions (Scenario 1)

Figures 5.124 and 5.125 show the comparisons of the prototype configuration and the two optimal design costs over five years. These figures highlight the different rates of quadratic growth of costs between the three configurations which shows the potential for cloud waste if optimizations are not calculated. The quadratic coefficient of the prototype is greater than 1 (1.36), while the quadratic coefficients of the 3D and Cost/Perf optimizations are less than 1 (0.6 & 0.5, respectively).

Table 5.23 shows the system costs and the cost savings comparisons between the optimal designs and the prototype for both scenarios. In this table we included the total prediction intervals we calculated using a summation of the meta-algorithm forecast prediction intervals and the UM

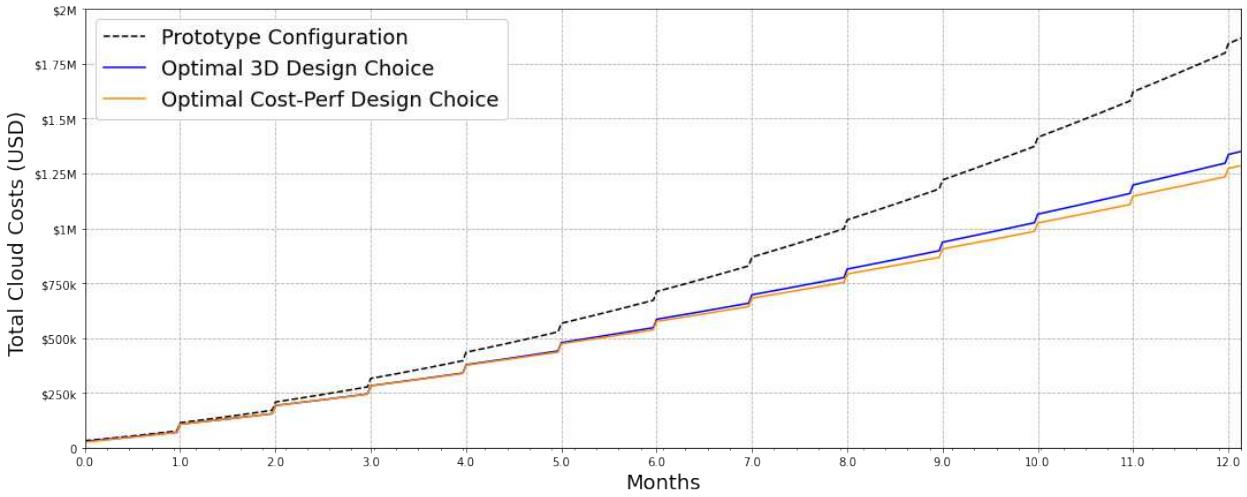


Figure 5.123: Prototype vs 3D and Cost/Perf Pareto-Optimal Solutions (Scenario 2)

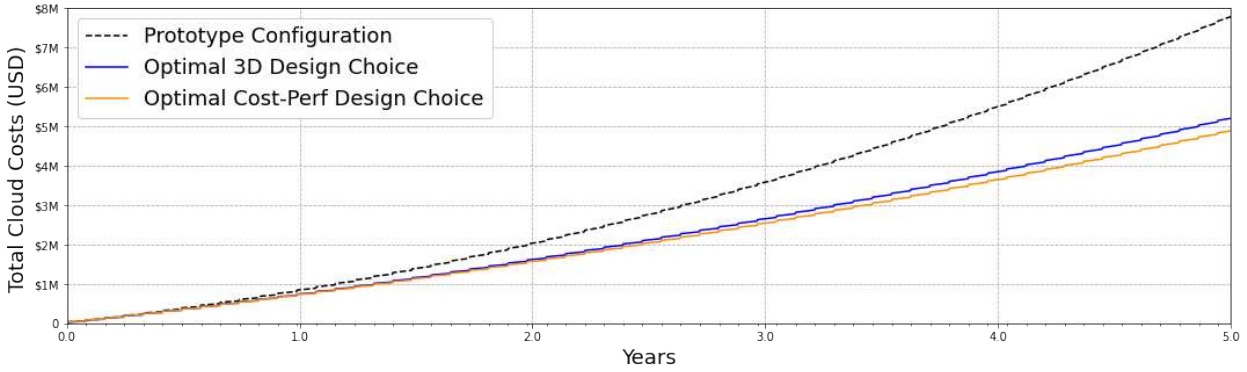


Figure 5.124: Prototype vs 3D and Cost/Perf Pareto-Optimal Solutions (Scenario 1)

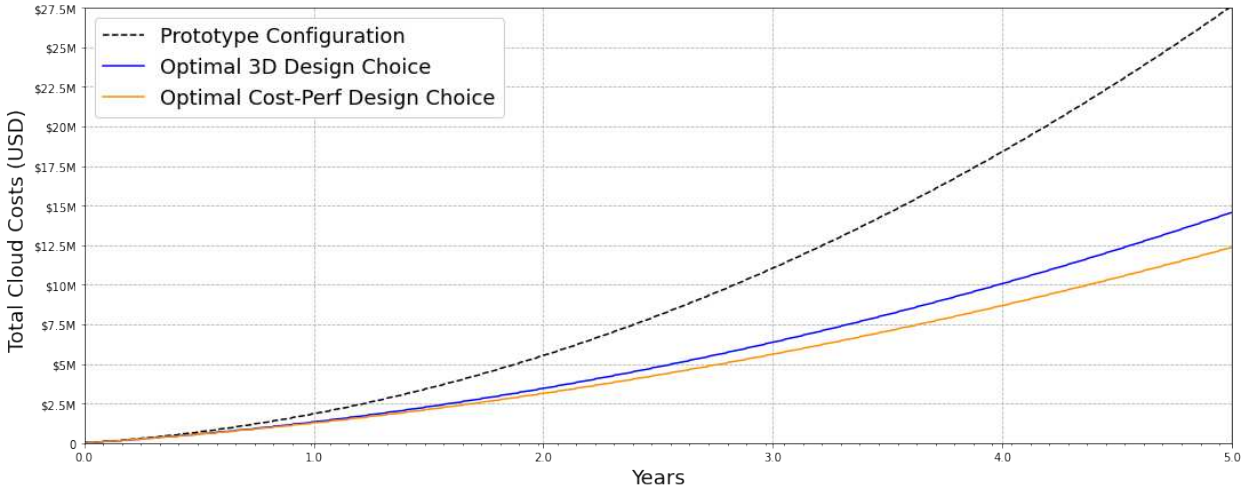


Figure 5.125: Prototype vs 3D and Cost/Perf Pareto-Optimal Solutions (Scenario 2)

simulation V&V RMSE ($864.6) from Table 5.13. The UM V&V simulation residuals were not normally distributed so we can not use their RMSE for our RMSFE directly. However, we assume these skewed distribution residuals are relatively similar between simulations, therefore using the V&V RMSE from Table 5.13 provides a relative prediction interval. Calculating the simulation differences provide a relative cost savings useful for insight into the value of YAMM. In addition to using the V&V RMSE for the relative prediction interval we also have the following assumptions about the cost savings in Table 5.23.

1 The two scenarios we simulated are less stochastic the prototype empirical data which had many known deficiencies, therefore this would reduce S3 errors in our estimations
2 Our scenario production environment would reduce EC2 cost errors with a reduction of software versions
3 Our scenarios don't have other VPCs running in parallel which would reduce NetFw errors
4 Relative differences between 2 simulations represent relative differences to real costs
5 No changes in cloud cost rates over time
6 No changes to scenario data processing throughput
7 We did not account for inflation

The estimated cost of TVD is the difference between the 3D Pareto-optimal design and the cost/performance optimal design. For scenario 1 this cost is $10k and $320k over a 1 and 5 year period, respectively. For scenario 2 this cost is $64k and $2.2M over a 1 and 5 year period, respectively. These costs are useful to include in the analysis and design of data retention policies.

When we take the cost of executing YAMM into account, we are able to calculate the ROI for this prototype system. It took roughly 1,400 man hours for the SE and AA teams to execute their portions of YAMM. For the sake of this calculation, we're assuming the other portions of YAMM would be normal activities for a software development project. For this relatively simple data system prototype, we had roughly 50 man hours per week for the first 3 months for gathering requirements and system architecture design. After this initial requirements and architecture phase,

Table 5.23: Optimization Savings Forecast for 1 - 5 Years

Simulation Config	Total System Cost	Cost Savings Compared to Prototype	Percent Savings Compared to Prototype
Scenario 1	**(±$2,176)**	**(±$4,353)**	
1 Year			
Prototype	$849,400	-	-
Optimal 3D Design	$746,297	$103,104	12.1%
Optimal Cost/Perf	$736,196	$113,205	13.3%
5 Years			
Prototype	$7,796,534	-	-
Optimal 3D Design	$5,208,359	$2,588,175	33.2%
Optimal Cost/Perf	$4,888,341	$2,908,193	37.3%
Scenario 2	**(±$3,711)**	**(±$7,422)**	
1 Year			
Prototype	$1,866,793	-	-
Optimal 3D Design	$1,350,370	$516,423	27.7%
Optimal Cost/Perf	$1,285,658	$581,135	31.1%
5 Years			
Prototype	$27,714,818	-	-
Optimal 3D Design	$14,619,614	$13,095,204	47.2%
Optimal Cost/Perf	$12,406,376	$15,308,442	55.2%

the SE and AA teams averaged less than 15 hours a week for rest of the 18 months. If we assume $250 per hour for their time, then we can calculate an estimated ROI for this case study.

Figures 5.126 and 5.127 show the man hour costs combined with the cost savings of implementing the 3D and cost & performance optimal systems. Positive ROI for scenario 1 happens at 777 and 742 days from the start for the 3D and cost/perf optimization settings, respectively. Whereas, for scenario 2, the ROI is realized much sooner at 361 and 344 days for the 3D and cost/perf optimizations, respectively. The differences in the 2 scenarios provide context for the savings potential for the average cloud system budget which is $12 million or more annually compared to scenario 2 which was roughly $2.5 million the first year.

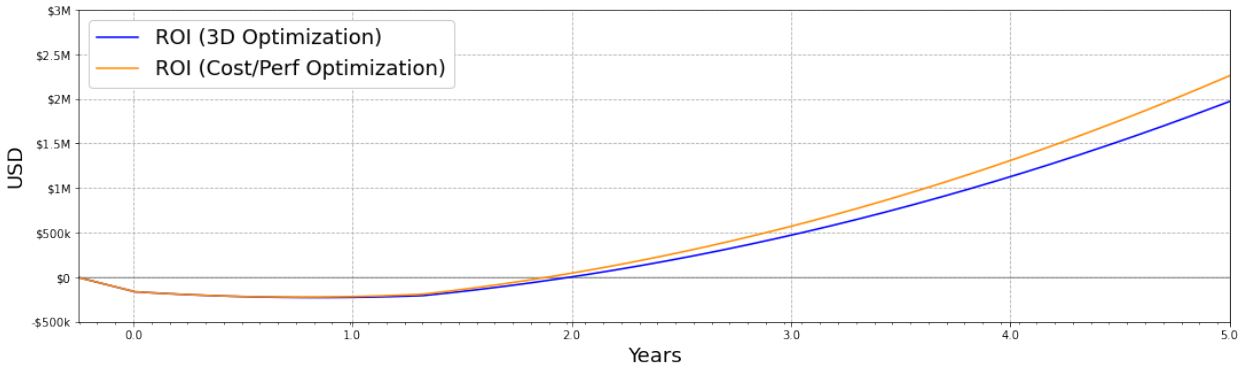


Figure 5.126: ROI for 3D and Cost/Perf Pareto-Optimal Solutions (Scenario 1)

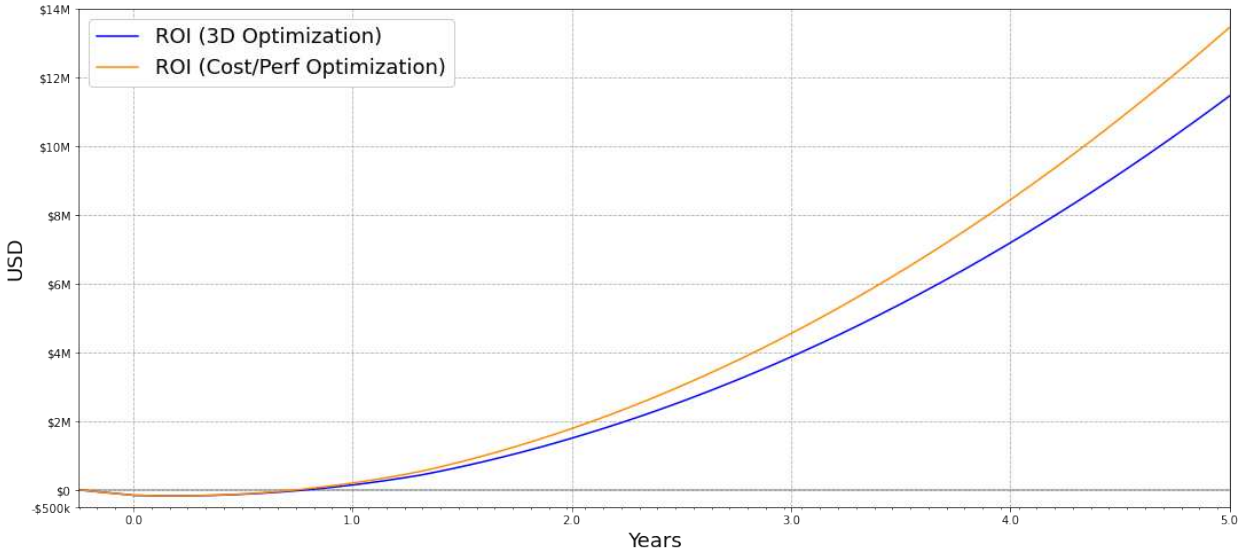


Figure 5.127: ROI for 3D and Cost/Perf Pareto-Optimal Solutions (Scenario 2)

5.8 Hybrid Cloud Conclusion

In this section we first provide a summary of the hybrid cloud case study chapter then provide additional details of the conclusions about the results, the challenges and limitations of the case study, and finally future work that could provide additional capabilities for the case study. This hybrid cloud case study in this chapter offered a demonstration of YAMM implemented towards the design, development, deployment, and optimization of a real-world data system prototype. During this case study the team completed 4 MVPs, over 4 YAMM iterations, 5 UM versions, and 2 optimization cycles over 21 months.

Initially, this hybrid cloud case study served as the foundation for aggregating and refining our MBSE, data analytics, DevOps, and FinOps guiding principles that coalesced into our development of YAMM and the UMA. This chapter provided, not only the origins of YAMM, but also a demonstration of the simulation and optimization capabilities of the final UM built for designing, developing, deploying, and optimizing a real-world data system prototype. We demonstrated how the UMA successfully creates modular and reusable modeling libraries which significantly reduced future modeling efforts and time. The case study showed how YAMM enables critical system analysis and optimization capabilities through the collections of automated resource usage and metadata and continuous system specific feedback to enhance improvements. The results showed a positive ROI and other qualitative values provided through the use of YAMM and the UMA. Unfortunately, limitations with the computational performance brought on by the tools and modeling environment restricted the optimization and number of simulations.

5.8.1 Unified Modeling Approach

The successful modularity and reusability of the hybrid cloud modeling libraries was demonstrated several times by Ver.1 libraries reused and upgraded through the models up to Ver.5. The modeling time for each new model version was significantly decreased compared to the previous. The modular nature of the libraries qualitatively made it easier to upgrade simulation capabilities by focusing on the individual activities and opaque behaviors rather than the entire model. The best demonstration of this happened between Ver.4 and Ver.5 of the models which represented the largest upgrade in capabilities, but took the least amount of time.

The ability of our UM to accept scalar or time-series data input enabled us to capture manual and time-dependent system behaviors not possible without a more complex model. These time-series array inputs also enabled us to validate the model against empirical data. Our use of three different model accuracy measurements in the UMA provided useful insights into origins and type of modeling errors seen in the resource usage and cost predictions. Measuring percent error, RMSE, and residual entropy enabled easier troubleshooting which saved us time. Finally, the UM

compatibility with a common analysis environment like python enabled us to leverage its powerful analytical and optimization packages.

5.8.2 YAMM

This section summarizes quantitative and qualitative values of YAMM, quantifies the ROI for this case study, and lists the hybrid cloud UM accuracies and speed of simulation. The case study in this chapter provided a practical demonstration of YAMM enabling the UMA towards empirically-based and accurate resource and cost estimations which provide accurate system optimizations and analyses. Data collection from the test and operational system environments are the cornerstone of enabling these accurate simulations. In addition to the data collection, YAMM also provides a methodology that integrates well with the cadence of DevOps by incrementally and iteratively modeling, designing, and optimizing software-centric systems. This incremental and iterative cadence is necessary for these types of systems as they evolve with the fluctuating system requirements often seen in the software industry.

While the Harmony aMBSE provided the framework, our key MBSE capabilities provided the refinements and extensions for YAMM to successfully design, analyze, and optimize the hybrid cloud system. The results in this chapter showed that the UMA which was enabled by YAMM successfully showed the following capabilities designed, analyzed, and optimized our software-centric data system prototype.

- Verify and validate MBSE model functional specifications using simulation
- Accurately capture software-centric data system emergent behavior using time-dependent simulation
- Automate system optimization and perform Pareto and sensitivity analysis
- Automatically collect operational and test data to develop algorithms and validate system models
- Provide continuous feedback of system specific knowledge improve the design and operations throughout the system's life cycle

5.8.2.1 Quantitative ROI

For our prototype system we calculated the cost savings using Pareto-optimal analysis for cost, performance, and TVD objective functions. Results show the total savings over 5 years compared to our default prototype system was $2.2M for scenario 1 processing and storing 20 TiB a week vs a total savings of $12.7M for scenario 2 which processed and stored 100 TiB/week. For context, the non-optimal prototype system 1 and 5 year costs were $850k and $7.8M respectively.

When using Pareto-optimal analysis for only cost and performance objective functions we saved more money, however it was at the expense of deleting the raw data. In this design the total savings were $2.5M and $15M over 5 year for scenarios 1 and 2, respectively. The time to recoup YAMM costs happened at 742 and 344 for scenarios 1 and 2, respectively. The cost of the TVD and keeping the raw data in the cloud archived for 5 years was $320k and $2.2M for scenarios 1 and 2, respectively.

As we suspected, the number of days to start generating a positive ROI was dependent on the use case, the size and rate of system data ingest, the organization's objective functions, and hourly pay of the team implementing YAMM. For instance, it took 777 days for scenario 1 to recoup the YAMM costs vs 361 days to recoup costs for scenario 2. Our cost calculations assumed the YAMM team spent 50 man-hours/week for the first 3 months before the system is deployed, then spent 12.5 man-hours/week for 15 months while they collected resource data, created algorithms, and updated the UM. We also assumed each man-hour cost the company $250/hr.

In addition to the Pareto-optimal analysis cost savings, we found savings while analyzing the cost and resource data of the prototype. The results showed YAMM's in-depth resource usage analytics and data exploration highlighted costly processes that were not shut down properly. These processes going un-noticed would have cost roughly $8k/month. This is a small savings, but shows an example cost benefit of analyzing cost data carefully.

5.8.2.2 Qualitative Value

The results from the Pareto, fuzzy Pareto, and sensitivity analyses provided the system engineers with an increased understanding of the prototype's dominant and strong system charac-

teristics. This has been shown to lead to more robust system designs and additional savings in design time. Additionally, the sensitivity analysis showed that these dominate system characteristics continued to be optimal choices in scenario 2 where the data input increases by a factor of 5. This knowledge could save on potential upfront costs from those stakeholders who may try to over-design the system to be future proof.

The application of YAMM to our prototype resulted in meeting our prototype objectives. We identified the specific data format (format 3) with its specific recording format, communication protocol, and ETL storage strategy that resulted in the highest performance and lowest processing cost per GiB. From a qualitative value, identifying these positive system attributes helps system engineers derive future requirements and standards which can save time and money on future development and integration costs.

Results from the simulations provided cloud compute, memory, and network usage which can be used to estimate on-prem CapEx and OpEx costs towards a larger trade study balancing cloud vs on-prem costs. This is another optimization we could do with YAMM to further optimize our data system.

5.8.2.3 Cloud Model Accuracy and Speed

Table 5.24 shows a summary of the storage, compute, network, and total cloud resource cost estimation accuracies from the cloud case study. The results show that our cost predictions as a function of time had an RMSE of $865 in context of a system cost of $361k over a 4.5 months validation window.

Our cost estimation percent error was 17.1% which was only slightly lower than the average cloud budget overrun seen by the Flexera report [13]. However, the benefit of our framework is that it already identified specific areas of simulation improvements. For instance, using storage cost residuals and correlating them to the daily ETL pipeline data processing we found that the majority of the 9.8% error is associated with our lack of modeling CRUD costs in the cloud. Additionally, we found the majority of the network firewall residuals do not correlate with any other resource cost or usage and determined these cost error can be attributed to other teams' network usage. To

quantify the randomness of our simulation errors results showed that our model had 3.1 residual entropy out of a maximum of 4.6. This indicates we have non-random behaviors that we need to fix. This supports the residual errors we identified with the data storage and network costs.

Table 5.24: UM Accuracy of Resource Cost Estimations

Cloud Resource Costs	Percent Error	RMSE	Residual Entropy	AWS Total	Sim Total
Data Storage Cost	9.8%	$90.0	3.5	$75,839.57	$68,437.75
ETL Compute Cost	14.8%	$253.35	3.4	$108,504.41	$92,477.50
Network/Firewall Costs	21.6%	$747.14	2.4	$176,910.80	$138,643.55
Total Cloud Costs	17.1%	$864.60	3.1	$361,254.79	$299,558.80
Max Entropy: 4.6					

The hybrid cloud case study UM project had over 155,000 elements including the model libraries. The average execution time of the simulation for 127 time steps was 13 minutes. This is a relatively slow execution time which we believe is associated with the limitations of the server resources available to us, the tools, and the language. For instance, these simulations took 3 minutes on server resources that had twice as many resources as the simulations that took 13 minutes. However, the larger resources still had their limit of 300 time steps before they no longer ran. This indicates a larger issue with resource allocation in the tool or with the languages we used.

5.8.3 Challenges and Limitations

The demonstration of our novel YAMM and UMA in this chapter successfully showed their capabilities, however we provide the following list of the challenges and limitations we documented through this case study.

1 The cost savings we report in the results are in context of the possible savings from the prototype cloud configuration. There was no prior optimization or cost savings data retention considerations during the prototype phase since these were not our number one priorities at the time. The magnitude of these savings may not be typical for legacy systems that

have already implemented some cost savings procedures. However, according to the Flexera study [13] 49% of organizations don't have a team to advise on, manage, or execute cloud cost optimization strategies. The organizations without an optimization strategy could save millions of dollars a year.

2 We did not implement the SRA during this case study. This case study was a prototype and didn't last long enough to make it worth implementing in the model. Implementing the SRA is recommended for projects lasting longer than 2 years.

3 We did not realize the difficulty in identifying resources to specific compute processes. If we would have specified tagging each compute job the SW-Dev team could have made this identification process easy enough to automate. This would have saved us time and most likely made the simulation more accurate.

4 We did not build a data collection framework around any of the tests run by the developers in the DevSecOps environment. Collecting data from this stage was something we realized we should have done in hindsight. This is why there is no empirical feedback from the testing environment. Data earlier in the prototype time line could have saved 2-3 months on the optimization steps.

5 Our final Ver.5 model included the staging bucket in the same AWS account as the ETL scripts for cost accounting and a reduction of network firewall costs. Network firewall charges were thousands of dollars on days with large data transfers. We should have accounted for this cost reduction in the model, however we did not have any data to validate our assumption that it would not have increased other costs so we did not account for this in our total savings. This implies the cost savings could have been higher than we forecast.

6 One downside to using the Cameo tool suite was the lack of openness to the model libraries. These are not easily shareable with other modeling tool suites. While, there are translations of these models to other tools, we have found that there are functionalities that break in the

libraries if this is done. This appears to be a reflection of the current lack of non-proprietary SysML tools and not the modeling approach.

7 The simulation viewpoint works well for the visual simulation elements, however the tool suite we chose did not enable easy viewable access to the source code of the opaque behaviors, functions, and utilities. This limitation extended to version control of the source code which was didn't have any way to see the differences between versions. Replicating Git version control structure should be default capability in any software related tool.

8 The modeling tool we chose was not well suited for this type of simulation and consumed significantly more RAM than we initially assumed. This resulted in poor simulation computational performance. This was most likely a combination of the tool, languages, and the VM it was hosted on. An example of this was the quadratic increase in simulation execution time as a function of simulation time steps. For our largest SysML model, after roughly 200 time steps the simulation times-out due to insufficient RAM. Finding a more efficient modeling tool may solve this issue, otherwise increasing computational resources may be needed.

9 The modeling tool and languages used for the cloud case study were not ideal choices to implement the unified modeling approach. This led to many non-standard software solutions that resulted in a less robust simulation model. In the next section we discuss future work to explore tools and languages that are better suited for our approach.

10 The Pareto-Optimal design we chose is based on the fixed number of time steps limited by the tool and our current resources. The optimal design would most likely have changed depending on increasing the number of time steps. Our extrapolations for cost savings were limited to these fixed settings. An example of this is setting the t_{TVD_o} to 180 even though our simulation only lasted 126 days. We most likely would have seen a different set of TVD optimal settings that only retained the archival data for the t_{TVD_o} amount of time. Anything longer than this would have cost too much. While this TVD objective function was just

an example, it is important to consider these effects of any objective function built. To compensate for this limitation, we showed the optimal design without TVD for context.

11 Using two objective functions may be easier to define, but may you miss a more optimal design choice compared to adding more. Additional objective functions, even if the impact is small, may provide a more optimal or robust system. However, poorly or incorrectly defined objective functions present risk to choosing a sub-optimal design if the analysts don't interpret the results correctly. The presence of data format 2 on the Pareto surfaces is a good example of this. It's performance was so poor that it caused an increase to the TVD score large enough to be on the Pareto surface. However, in terms of performance and cost it was the worst data format to choose and would have resulted in fairly little cost savings over the prototype.

5.8.4 Future Work

In this section we describe the future work for the cloud case study that could provide additional research and production value. Our current goals for this case study include decreasing simulation execution time, increase the accuracy of the model, and increasing the size and complexity of the model. Additionally, we see value in training ML models on the resource usage and cost data and use these models in the UM for increased accuracy and the ability to automate the process easier. This would enable us to continually ingest changes to the system and ensure it is still optimally configured.

To further explore large hybrid cloud data system future work will include the optimization of the entire hybrid cloud system. This would involve analyzing the right balance of on-prem hardware and cloud-based computing and storage. For example, what is the right balance of on-prem hardware and software to cloud hosted software and storage? Furthermore, cost-benefit analysis between different cloud providers, each with different cost models, for data systems is on the future road map to reduce organizations cloud costs using YAMM and the UMA.

Finally, the exploration of implementing the UMA in SysML version 2 is were we believe we can realize much of the speed up in simulation and optimization performance. It also provides the opportunity to utilize our model libraries with more tools which would provide additional value than they currently have.

Chapter 6

Conclusions

In this chapter we present our conclusions about our novel contributions, YAMM and the UMA, towards defining an MBSE methodology and modeling approach for software-centric data systems. We summarize the conclusions and novel contribution from each case study, answer our research questions, discuss the challenges and limitations, and summarize the future work. During our research we successfully demonstrated YAMM being applied to an OLTP and an OLAP data system using an aircraft sensor system and hybrid cloud system as the respective case studies.

Our novel contributions include a new MBSE methodology and modeling approach focused on the design, development, and sustainment of software-centric data systems towards the analysis and optimization of system objectives such as resource reduction, increased performance, and reduced costs. Results from our research and our case studies demonstrate that YAMM in combination with our UMA can reduce costs, increase performance, and provide useful insights into critical system characteristics for the design, development, and sustainment of both OLTP and OLAP type data systems. Together, YAMM and the UMA integrate MBSE with DevOps and FinOps principles to improve data system performance and reduce costs through continuous and empirically-driven feedback.

Sections 6.1 and 6.2 provide additional details about the conclusions from each case study. Section 6.3 addresses our research question about the attributes of a MBSE methodology and modeling approach that can provide a positive ROI. Section 6.4 summarizes the qualitative values of these this methodology and modeling approach and Section 6.5 provides the performance and accuracy of an MBSE model that unifies the simulation and specifications into a single model. Section 6.6 discusses the challenges and limitations of YAMM and our UMA. Lastly, Section 6.7 proposes future work that could advance the capabilities we developed in our research.

6.1 Legacy Aircraft Case Study

The aircraft case study focused on the addition of operational resource usage data feedback to select and analyze sensor fusion DL models. While the entire YAMM life cycle was not demonstrated, the focus of this case study was the integration of an aircraft sensor system SysML model, DL Model Cards, and the data analytics environment. This integration enabled us to select the best sensor fusion DL model based on performance, accuracy, and resource usage, rather than performance and accuracy alone. We demonstrated the advantages of YAMM's principles applied towards automating the ML training, selection, and analysis processes to keep up with the rapid advances in ML and sensors being deployed to DoD aircraft.

In addition to YAMM, our novel contributions within this case study were the extension of ML Model Cards and the creation of a SysML Model Card to better enable automated analysis and selection of the sensor fusion DL model. We extended the ML Model Card definition for use by our ML analysis by adding the following data:

1 Compute, volatile memory, non-volatile memory, and networking resource usage for system analysis.

2 Benchmark and generalization MSE to include accuracy in the system analysis.

3 Pre-processing and model inference latencies to include performance in the system analysis.

4 Links and information about the source of ML training, testing, and validation data to increase trust in the model.

Many resource-limited OLTP systems have timing constraints and need to consider both performance and resource usage when developing new software or selecting ML models to run on the system. Additionally, ML algorithm development and performance will continue to accelerate and drive the need to provide a scalable and automated system analysis process.

We demonstrated the value of implementing YAMM's DevOps capability which enables automation and collaboration between system engineers, software developers, and the operations teams. Additionally, YAMM enables the collection of downstream operational resource data towards automation of trade study and system analyses for selecting ML models or new software.

The results of our GD-MAGS ML selection and system analysis approach also showed that the automation of rapid ML or software deployment to an OLTP data system can benefit from standardized, machine-readable, and quantitative ML Model Cards, and machine-readable SysML system model specifications. Therefore, YAMM and our other novel contributions can provide the enabling technologies that support rapid iterations and incremental improvements of OLTP data system design, system analysis, and software or ML development.

6.2 Hybrid Cloud Case Study

The cloud case study focused on the optimization of cost, performance, and the time value of data for a prototype hybrid cloud OLAP data system supporting an example aircraft system. The goals of the study were to reduce cloud costs, increase system performance, and define a data retention policy. This case study demonstrated the implementation of YAMM towards the design, development, deployment, and optimization of a real-world Data Mesh prototype. During this case study the team completed 4 MVPs, over 4 YAMM iterations, 5 UM versions, and 2 optimization cycles over 21 months.

This case study demonstrated the novel ability of YAMM to successfully aggregate MBSE, data analytics, DevOps, and FinOps guiding principles into a single methodology refined for the modeling, design, analysis, optimization, development, and sustainment of software-centric data systems. Additionally, this case study demonstrated the novel aspects of our UMA that are enabled by YAMM's collection of operational and system test resource and metadata.

We demonstrated how the UMA successfully creates modular and reusable modeling libraries, which significantly reduced future modeling efforts and time. The case study showed how YAMM enables critical system analysis and optimization capabilities through the collections of automated resource usage and metadata and continuous system specific feedback to enhance improvements. Results from the comparison of our AWS cloud system prototype showed a positive ROI and other qualitative values as a result of YAMM and the UMA.

The specific capabilities we added to the hybrid cloud UM were focused towards several objectives for the case study data system. The main objectives were to reduce costs, increase performance, and develop cloud governance policies. To meet these case study objectives, we defined and accomplished the following specific goals:

1 Identify the best recording format and communication protocols between three example aircraft formats.

2 Identify the optimal ETL pipeline storage utilization strategy between EBS and S3 for the three formats.

3 Define the optimal data retention and data archive policies.

4 Estimate on-prem CapEx and OpEx costs for the hybrid cloud system.

After validating our model, we ran a matrix of simulations and found the Pareto-optimal system from these. We compared the Pareto-optimal system simulation to the non-optimal prototype simulation to estimate the time period to realize a positive YAMM ROI, the cost savings over 5 years, and any system performance change. Our system simulation and scenario ROI assumptions included 3 months of initial design and 15 months of incremental updates to the model.

Our results showed a positive ROI after 777 daysand a. 5 year savings of $2.2M without losing system performance. Therefore, implementing YAMM and the UMA has shown a positive ROI, savings of millions of dollars, increases in system performance, and insight into defining data policies. This methodology also adopts the DevOps principles of rapid iterations and incremental improvements towards the design, analysis, and optimization of data systems. Therefore, our novel methodology and modeling approach can also save costs throughout a system's acquisition and sustainment life cycle phases.

6.3 Key Attributes

In this section we answer the first question that began our research into the value of MBSE methodologies and modeling approaches. We restate our first research question here: *What are the*

key MBSE methodology and modeling attributes that result in a positive ROI during the design, development, and sustainment of a software-centric data system?

Our results showed that YAMM can save millions of dollars a year while improving or maintaining performance on software-centric data systems while balancing other functional objectives. The magnitude of cost savings and the length of time to reach a positive ROI are dependent on the data system's use cases and functionality, the data system size and complexity, users' demands on system resources, the organization's objective, and the cost of the team implementing YAMM.

For the two scenarios we analyzed in cloud case study, results showed a positive ROI being realized for the system between 1 and 2 years from the start of the design and through system deployment. After 5 years the cost savings for 2 example scenarios was between $2.2M and $12.7M. These savings are in context of our prototype system with estimated five year costs of $7.8M and $27.7M for these scenarios, respectively. The savings represent a 28% and 46% percent cost reductions.

The following lists of MBSE methodology and modeling approach attributes are contained within YAMM and the UMA which produced the positive ROI and cost savings we stated above for the design, development, and sustainment of software-centric data systems.

YAMM Methodology Attributes

1 Collect operational and system test resource usage and performance data towards validation and increased accuracy of system models and system analyses.

2 Perform system optimization, Pareto and fuzzy Pareto studies, resource analyses, and sensitivity analyses.

3 Provide continuous feedback of system specific knowledge to optimize the design and operations throughout the system's life cycle.

UMA Modeling Attributes:

1 Create semantically consistent system specification models to increase model reliability

2 Visualize complex system models with intuitive, graphical representations to increase human interpretation accuracy.

3 Merge specifications and simulations into a single, unified model to ensure consistency between specification and simulation behaviors.

4 Accurately capture emergent system behaviors using time-dependent simulation.

5 Utilize modular and reusable model libraries to reduce modeling and design time.

6 Automatically verify MBSE model functional specifications using simulation.

7 Automate validation of the MBSE model specifications, simulations, and tool set.

8 Standardize an open model simulation interface to integrate with standard data analytics environments.

6.4 Qualitative Values

In this section we discuss the qualitative values as a follow-on to the initial research question in the previous section. We restate this follow-on research question here: *What other qualitative values does an MBSE methodology and model with these attributes provide?*

In addition to the positive ROI, our novel YAMM and UMA produce additional unmeasured quantitative and qualitative values for data system organizations. YAMM's continual empirically-derived updates to the UM ensures that it remains accurate throughout the life of the data system. This enables organizations to re-evaluate the system's optimality periodically which is necessary as the system's usage changes over time and cost savings and performance are dependent on these use cases.

Results from both case studies showed that models with standard interfaces enable integration with our python-based analytics environment which allowed us to perform Pareto and fuzzy Pareto analysis, sensitivity studies, Monte-Carlo analysis, meta-algorithm development, ML classification, and any other analytical capabilities needed to analyze and optimize complex systems. With the UM compatibility with common analytical environments, users can choose the category of mathematical optimization or analysis that works best for their system, resources, and use cases. The identification of dominant characteristics and the ability to optimize systems can lead to better and more robust designs.

YAMM's prescriptive nature for data systems better enables non-MBSE experts by reducing their requirement to refine and extend a more generalized methodology. Our experience in this field shows that this is valuable to new MBSE practitioners and lowers the effort to build valuable models. This prescriptiveness also reduces the work required by MBSE experts to apply to their data system.

Unlike our observations with many DoD MBSE models and recent publications, the application of YAMM is meant to continue to use the system specification models after the system has been built. For instance, YAMM can be used as an organization's FinOps methodology for data systems. This enables the reuse of the specification and simulation models when transitioning to the life cycle sustainment stage undergoing continual upgrades and patches that many systems operate in for decades. This methodology, much like FinOps, brings together the financial, engineering, development, and operations departments to manage and optimize cloud spending. The benefit of using a visual specification and simulation language in the UMA provides these departments a more comprehensive understanding of the data system to ensure the decision makers are well informed. Graphical and semantically consistent system specifications can result in systems that are more likely to have fewer issues with system integration which also saves time and money.

YAMM provides a continuous feedback of knowledge which, in reference to the Blanchard and Fabrycky 'System-Specific Knowledge' curve, in Figure 1.1, shifts system knowledge to the left in order to affect system change while it is still relatively easy and incurred costs are lower. To further reduce cost YAMM is a use case based design technique built to incrementally and iteratively design and optimize the system. This MVP approach provides flexibility for requirements changes throughout the design and other life cycles of the system. By focusing on the next valuable deliverable design, rather than design the entire system upfront, rework is reduced when requirements change due to new system specific information or new customer needs are realized.

We also demonstrated that the UMA met the FEMMP evaluation criteria [1] for scalability, simulation, checking, and reusability of our methodology, which is a positive indication of the quality of the methodology. We demonstrated the verification of the UM functional specifications

by testing specific logical conditions with the simulation results. This has been shown to increase the functional accuracy and quality of its system specifications [105, 106]. This capability aligns with with the FEMMP "checking" category.

A modular unified model with a reusable set of modeling libraries enables faster future expansion in size or complexity of the model which can save system engineers and simulation teams time and money. This aligns with the FEMMP "scalability & reusability" evaluation criteria. Additionally, our UMA produces a model that accepts both scalar data and time-series arrays as input can estimate real-world, unique, and time-dependent conditions. This can be used for early and continuous empirical feedback which can increase the rate of system-specific knowledge early and throughout the system's life-cycle [5]. These capabilities align with the FEMMP "scalability, simulation, & reusability" evaluation criteria.

6.5 Unified Model Accuracy and Speed

This section presents the results from the final research question about quantitative accuracy and performance of a model built using the MBSE modeling approach that have the attributes mentioned our first research question. We restate our final research question here: *What is the accuracy and speed of a SysML-based simulation-enabled model that estimates system resource usage and costs as a function of time?*

The cloud case study showed that the UMA created an accurate, time-dependent, simulation-enabled, and verified and validated system specification UMs capable of capturing emergent system behavior. These capabilities and attributes were enabled by YAMM through the collection of empirical resource usage and cost data from the operational data environment through the development and use of the system.

The hybrid cloud case study results show that our cost predictions as a function of time had an RMSE of $865 in context of a system cost of $361k over a 4.5 months validation window. The simulation's percent error in cost was 17.1% which was lower than the average cloud budget overrun seen by the Flexera report [13]. However, the benefit of our framework is that it already helped

us identify specific areas of simulation improvements through the use of different model accuracy measurements. The UMA measures model component accuracies with percent error, RMSE, and residual entropy to provide a clear understanding the nature of the errors when validated against empirical data. The ability to identify individual components' residuals enabled us to identify the source of the model errors quickly. Using residual entropy also enables modelers to know when to stop modeling since the residuals will be associated with a normal distribution and there is nothing but noise to model. Using these capabilities in the cloud case study we identified the largest errors in total cost were the lack of CRUD cost modeling and lack of correction for network firewall costs associated with a different account.

Our research results showed that the speed of the simulations using TWC's server-side simulation was dependent on the number of model elements, the length of the use case simulation, the number of time steps in the simulation, the execution tool efficacy, and the amount of compute resources available to the server. The hybrid cloud case study UM project had over 155,000 elements including the model libraries. The average execution time of the simulation for 127 time steps was 13 minutes. This is a relatively slow execution time which we believe is associated with the limitations of the server resources available to us, the tools, and the language. For instance, these simulations took 3 minutes on server resources that had twice as many resources as the simulations that took 13 minutes. However, the larger resources still had their limit of 300 time steps before they no longer ran. The limit on number of time steps indicates a larger issue with resource allocation in the tool or with the languages we used.

6.6 Challenges and Limitations

We developed the UMA with a focus on providing a practical and prescriptive approach towards implementing MBSE for software-centric data systems. The assumptions we made for this approach limit this modeling approach to data system or similar systems. Our key modeling and simulation assumption is that the simulation can be done completely inside of the UM language and tool set. This is a limitation where the UMA will not work well for system's that need physics-

based or entity-based simulations and analyses. These types of system simulations are best done outside of the system model in tools built for these specific purposes. There are other MBSE methodologies that address this.

Another limitation to the UMA is the lack of current production-ready tools and languages for a high performance simulation capable of timely system optimizations. Current production tools use a significant amount of compute resources and the simulations are slow. This indicates a lack of maturity for simulation capabilities in current tool suites. The SysML version 2 language and current tools have a promising outlook in the near term, but have not yet been proven for production quality models. A work around to this limitation is to increase the size of compute resources and parallel jobs running these simulations.

The collection of resource usage data from a non-cloud environment is a challenge and requires the use of third party or custom written software to record this information. These type of logging tools are common, however they also consume resources to run and record this data. Cloud providers collect daily data for free, however they charge a small service fee when collecting high fidelity data down to the minute or hour for some resources. These costs should be taken into account when optimizing the system.

Another challenge is tagging software processes appropriately to enable accurate classification of resources used by system components and behaviors. Without some software expertise, this could prove challenging for some organizations. A possible workaround includes separating the different sub-systems and software processes into different vNets, subscriptions, or using other methods to differentiate resource consumption without tagging software processes.

6.7 Future Work

Future work will include implementation and analyses of different graphical specification languages and simulation languages to reduce limitations seen with the tools and languages used in our case studies. SysML version 2 and many of the modern tool sets that support it seem promising. The advantages seen with decoupling the modeling tool from the visualization and model reposi-

tory may solve many of the issues and limitations we discussed in previous chapters. Furthermore, creating a Reference Architecture (RA) using our modular unified modeling libraries and design patterns developed and refined during the hybrid cloud case study could increase the speed of the design of future cloud based data systems. This could also enable new system engineers to adopt the YAMM and UMA faster.

The DoD and many other industries value system and cyber-security as high as, or higher than, their cost and performance objectives. Future research will explore the the capability to model and collect system security related resource usage and cost data within the YAMM framework. This would further increase the value of implementing YAMM and the UMA for data systems in the commercial and government industries. While YAMM already defines the mechanisms to adopt this capability, future research would need to explore the data collection, modeling, V&V, and mathematical security objective function definition to capture the core security attributes of; confidentiality, integrity, and availability.

Research into the applicability of YAMM on hardware-centric data systems or new types of systems would enable our novel capabilities for more industries. However, if YAMM becomes too generalized then it may lose part of the value in being practical and prescriptive for new users.

Acronyms

1553 MIL-STD-1553 33, 166

AA Analytics & Algorithmics 156–158, 160, 172, 199, 285, 286

AADL Architecture Analysis and Design Language 25

AC Aircraft 160

AD Anderson-Darling 282

ADS Agile Data System 152

AI Artificial Intelligence 5, 19, 39, 40, 45, 106, 150

ALH Action Language Helper 161, 176, 178, 193

AoA Analysis of Alternatives xi, xii, 107, 128, 130–132, 141, 142

API Application Programming Interface 6, 24, 33, 53, 90, 112, 141, 145, *Glossary:* API

ASOT Authoritative Source of Truth 15, 60, 112

ATO Authority to Operate 32, 46, 49, 157, 166, 184

AWS Amazon Web Service iii, ix, xiii, 12, 13, 30, 39, 157, 160, 169, 184–186, 208–210, 214, 218, 223, 225–227, 230–232, 234, 237, 242–246, 248, 249, 255, 292, 293, 299

BC Baseline Cost 232, 233, 235, 237

BDD Block Definition Diagram xi, 114–116

CapEx Capital Expense 13, 150, 239, 240, 250, 260, 261, 291, 300

CFD Computational Fluid Dynamics 35, 36, 54, 103

Ch10 IRIG 106 Chapter 10 166

CI/CD Continuous Integration & Continuous Delivery 8, 11

CNN Convolutional Neural Network 38, 39

CPU Central Processing Unit 39, 112, 204

CRUD Create, Read, Update, Delete 71, 103, 224, 246, 291, 305

csm cameo systems modeler 195

CSP Cloud Service Provider 30, 34, 98, 186, 209

DDS Delta Data Size 231, 234

Dev*Ops Development, [*], and Operations xi, 2, 11, 27, 86, 87, 141, *Glossary:* Dev*Ops

DevOps Development and Operations ii, iii, 1–3, 6, 8, 11, 16, 18, 25, 27, 42, 87, 88, 109, 112, 140, 143, 145–148, 288, 289, 297–300, *Glossary:* DevOps

DevSecOps Development, Security, and Operations ix, xii, 2, 18, 27, 42, 47, 50, 51, 86–89, 92–94, 97, 154, 156–158, 164–167, 184, 185, 208, 293, *Glossary:* DevSecOps

DI Digital Infrastructure ix, 42, 46–49, 66, 86, 88–93, 96, 97, 151, 154, 156–158, 161, 166, 169, 171, 184, 185, 207, 208, 211, 237, 240, 250, 258, 260, 261, *Glossary:* DI

DIRECT Dividing Rectangles 73

DL Deep Learning xii, 36–39, 103, 107, 110–114, 117, 118, 120, 122, 125, 127, 129, 132–136, 140–145, 147, 298

DoD Department of Defense 1, 5, 6, 11, 20, 26, 32, 49, 82, 149, 298, 303, 307

DODAF Department of Defense Architecture Framework 60

DP D'Agostino-Pearson 282

DPC Data Processing Cost 235

DS Dassault Systemes 161, 162, 175, 195

DSL Domain Specific Language 59, 161

DSOP DevSecOps Pipeline 47, 48, 88, 157–159, 164

DT Developmental Test ix, 19, 44, 94, 95, 98, 99

EBS Elastic Block Storage 13, 150, 227, 230, 234, 236, 239, 246, 300

EC2 Elastic Compute Cloud xiv, 39, 227, 229–237, 245–247, 254, 257, 285

EC2-Other EC2-Other Services xiii, xiv, 227, 229–233, 235, 236, 245–247

ESEM Executable System Engineering Method 25, 32

ETL Extract, Transform, and Load ix, xii–xiv, 13, 150, 154, 158–160, 164–166, 172, 174, 175, 178, 180–182, 184–187, 190, 198, 202, 203, 207, 208, 211–214, 225–229, 231–235, 237, 239–244, 246, 252, 253, 257, 279, 291–293, 300

FEA Finite Element Analysis 54

FEMMP Framework for the Evaluation of MBSE Methodologies for Practioners ix, 21–24, 54–56, 303, 304

FinOps Financial Operations ii, iii, 1, 11, 15, 16, 28, 50, 288, 297, 299, 303

fUML Foundational Unified Modeling Language 18, 32, 161, 175

FWaaS Firewall as a Service 227

Gbps Gigabytes per second 180, 181, 185, 202, 211, 231, 254, 258

GD-MAGS Gradient Descent Multi-Algorithm Grid Search 107, 112–114, 118, 120, 122, 125, 127, 132, 133, 135–137, 140, 143, 147, 299

GiB Gibibytes 181, 209, 227, 229, 231, 234, 239, 245, 248, 249, 259, 291

GNC Guidance, Navigation, and Control ix, xi, 74, 76–81

GP General Purpose 230

GPU Graphical Processing Unit 39

Harmony aMBSE Harmony agile MBSE process iii, xi, 6–8, 10, 12, 13, 16, 26, 27, 42, 49, 112, 148, 289

HTAP Hybrid Transactional/Analytical Processing 20, 44, 45, 94, 97–99, 150

HW Hardware 89, 102

I/O Input/Output 230, 246

IaaS Infrastructure as a Service 31, 90

IaC Infrastructure as Code 90, 166

IBD Internal Block Diagram xi, xii, 114, 115, 120, 128, 151, 163, 168, 175, 239, 250

ICD Interface Control Document 153, 158, 159, 165–167, 184, 225, 254

IDC International Data Corporation 260

IOPS Input/Output Operations Per Second 230

IP Internet Protocol 227

IRL Integration Readiness Level ix, xi, 47, 76, 82–85, *Glossary:* IRL

IS Information System 2, 54

IT Information Technology 1, 2, 5, 27

K8s Kubernetes 166, 185, 207, 231, 233–235, 237

NASA National Aeronautics and Space Administration xi, 12, 33, 82, 83, 114, 115, 117, 122, 123, 132, 133, 140

NAT Network Address Translation 227

NetFw Network Firewall xiv, 227–229, 231, 233, 234, 245, 247–249, 285

OFV Objective Function Value ix, x, 253, 258, 259, 262–264, 266, 267, 270–272

OLAP Online Analytical Processing 20, 37, 44, 45, 94, 98, 99, 109, 149–151, 164, 297, 299

OLTP Online Transaction Processing 20, 37, 38, 44, 45, 94, 97, 99, 109, 146, 149, 150, 153, 160, 164, 171, 172, 180, 185, 186, 240, 297–299

OMG Object Management Group 32

OOD Object-Oriented Design 25, 26, 55

OOM Order of Magnitude 5, 223, 244, 245, 247, 266, 281

OOSEM Object Oriented Systems Engineering Method 25

OpEx Operating Expense 13, 150, 239, 240, 250, 260, 261, 291, 300

OSI Open Systems Interconnection 90

OSS Open Source Software 25

OT Operational Test ix, 11, 19, 44, 46, 94, 95, 98, 99

PaaS Platform as a Service 19, 90

PC Processing Cost 232–235

PSO Particle Swarm Optimization 74, 160, 187, 201–204, 255, 257, 258

PVI Pilot Vehicle Interface 166

QUICURB Quick Urban Industrial Complex 36

RA Reference Architecture 307

RAM Random-Access Memory 219, 220, 255, 294

RMSE Root Mean Squared Error iii, 13, 67, 68, 223, 244–247, 281, 282, 285, 288, 291, 304, 305

RMSFE Root Mean Squared Forecasting Error 282, 285

RNN Recurrent Neural Network 38–40, 111, 113, 115, 125, 128, 129, 132, 140, 141, 148

ROI Return On Investment ii, iii, xv, 3, 5, 6, 13, 14, 16, 150, 285–290, 297, 299–302

RTOS Real-Time Operating System 26

S3 Amazon Simple Storage Service xiii, xiv, 13, 150, 157, 160, 186, 209, 211, 214–218, 223–226, 233–236, 239, 243–246, 254, 285, 300

SaaS Software as a Service 19, 31, 90

SDI Software Defined Infrastructure 47, 48, 90–93, 95–97, 156, 157, 164, 166, 169, 180, 184, 207, 208, *Glossary:* SDI

SE Systems Engineering xi, 3, 6–8, 11, 16, 21, 22, 26, 27, 32, 42, 43, 46, 47, 49, 50, 56, 57, 59, 71–73, 86, 151, 156–160, 166, 168–171, 183, 185–187, 191, 192, 199, 200, 235, 237, 285, 286

SOA Service-Oriented Architecture 55

SoS Systems of Systems 18, 28, 33, 34, 69

SRA System Readiness Assessment 47, 50, 76, 82, 85, 86, 293, *Glossary:* SRA

SRL System Readiness Level 47, 82–85, *Glossary:* SRL

SUT System Under Test 88, 89

SW Software 42, 48, 57, 86, 95

SW-Dev Software Development 46, 47, 49, 66, 88, 91, 151, 156–160, 165, 166, 171, 184, 185, 191, 207, 208, 224, 293

SWEBOK Software Engineering Book of Knowledge 89

SysML Systems Modeling Language ii, xi, xii, 3, 14–16, 18, 22, 25, 26, 32–34, 58, 59, 69, 112–116, 120, 121, 127–129, 140, 141, 145, 148, 151, 161, 162, 166, 175, 193, 211, 294, 296, 298, 299, 304, 306

TCO Total Cost of Ownership 29, 30, 260

TDDesign Test Driven Design xi, 53, 55, 58, 63, 65, 66

TDDev Test Driven Development 53, 63, 66

TiB Tibibytes 236, 245, 290

TRA Technology Readiness Assessment 82, *Glossary:* TRA

TRL Technology Readiness Level ix, xi, 47, 76, 82–85, *Glossary:* TRL

TVD Time Value of Data xiv, 254–256, 262–264, 266–273, 275–280, 285, 290, 294, 295

TWC Teamwork Cloud 193, 220, 256, 305

UAF Unified Architecture Framework 60

UM Unified Model x, xii, 9–11, 13, 15, 46, 48, 49, 51–53, 55, 56, 58–61, 63–67, 70, 76, 82, 83, 87, 91–93, 95, 96, 99, 100, 102–104, 106, 151, 152, 154, 156–158, 160, 170–172, 179, 180, 183, 184, 186, 187, 195, 200–202, 204, 206, 208–212, 214, 217–220, 222–224, 234, 235, 237–239, 242, 243, 249, 250, 255, 280, 283, 285, 287–290, 292, 295, 299, 300, 302–305

UMA Unified Modeling Approach iii, xi, xii, 9, 10, 12–17, 51–58, 60–69, 71–73, 86, 93, 94, 103, 106, 151–153, 161, 162, 170, 172, 173, 176, 180, 181, 183, 187, 188, 192, 195, 200, 206, 210, 219, 221, 222, 224, 235, 238, 239, 243, 288, 289, 292, 295–297, 299–307

UML Unified Modeling Language 25, 58, 59

USAF United States Air Force 109, 161

V&V Verification & Validation xi, 6, 11, 46, 48, 53, 63, 65, 86, 94, 156, 160, 208–212, 214, 215, 218, 219, 221–223, 227, 235, 238, 243, 244, 255, 285, 307

VM Virtual Machine 30, 59, 102, 186, 227, 233, 256, 257, 262, 265, 267, 271, 272, 294

vNet Virtual Network 90, 306

VPC Virtual Private Cloud xiv, 227–229, 231, 233–235, 245, 247–249, 285

WAN Wide Area Network 185, 202

WBS Work Breakdown Structure ix, xii, 151, 154–161, 184, 226

YAML Yet Another Markup Language 115, 128

YAMM Yet Another MBSE Methodology iii, ix, xi, xii, 7–17, 42–46, 48–52, 54–56, 63, 66, 67, 69–71, 77, 79, 86, 87, 90, 93, 94, 96–99, 102–106, 109–115, 119, 127, 132, 140, 141, 143, 145–152, 154–162, 170, 171, 184–186, 199, 207, 208, 210, 214, 218, 219, 225, 226, 237, 238, 285, 287–292, 295, 297–304, 307, *Glossary:* YAMM

Glossary

API An Application Programming Interface (API) is a particular set of rules and specifications that a software program can follow to access and make use of the services and resources provided by another particular software program that implements that API. 6

Dev*Ops Dev*Ops extends the DevSecOps concept and integrates any other automation required by the process. An example of this would be automated certifications, safety, analysis, or optimizations added to the DevSecOps process. xi, 2

DevOps DevOps is a methodology that aims to establish collaboration between programmers and operators to automate the continuous delivery of new software to reduce the development life cycle and produce quality software. ii, 1

DevSecOps DevSecOps extends the DevOps concept and integrates security methods into the process. DevSecOps is a software development process where security is built in to ensure application confidentiality, integrity, and availability. ix, 2

DI The physical and software defined foundation that enables the exchange of digital information, products, and services. It includes data centers, networks, servers, and software. Components can include: On-Prem HW, Cloud, Fiber Infrastructure, Software, and Security ix, 42

IRL Describes the state of a given integration between system components and provides a baseline from which maturity is gauged and advancement defined ix, 47

MVP the most basic version of a product with just enough features to be usable by early customers, allowing developers to gather feedback and validate their product idea before investing heavily in further development ix, 7

SDI The software infrastructure layer, based on virtualization, that is the definition of technical infrastructure controlled by computer hardware, that is entirely under the control of software. 47

SRA The guidelines and process for assessing the SRL of a system component and rolling up the SRL to higher levels of the system 47

SRL Describes the state of a given system using TRL and IRL to provide a baseline from which system maturity is gauged and advancement defined 47

TRA The guidelines and process for assessing the TRL of a system component and rolling up the TRL to higher levels of the system 82

TRL Describes the state of a given technology and provides a baseline from which maturity is gauged and advancement defined ix, 47

YAMM An Agile and Empirically-based Unified-Modeling, Design, Analysis, and Optimization Methodology for Software-centric Data Systems iii, 7

Bibliography

[1] T. Weilkiens, A. Scheithauer, M. Di Maio, and N. Klusmann. Evaluating and comparing MBSE methodologies for practitioners. *ISSE 2016 - 2016 International Symposium on Systems Engineering - Proceedings Papers*, pages 1–8, 2016.

[2] Bruce Cameron. Chapter 15 Reasoning about Architectural Tradespaces. In *System Architecture: Strategy and Product Development for Complex Systems*. Prentice Hall, 2015.

[3] Muhammad Azeem Akbar, Kari Smolander, Sajjad Mahmood, and Ahmed Alsanad. Toward successful DevSecOps in software development organizations: A decision-making framework. *Information and Software Technology*, 147:106894, 2022.

[4] Hironori Washizaki. *Guide to the Software Engineering Body of Knowledge v4.0*. 2024.

[5] Benjamin S. Blanchard and Wolter J. Fabrycky. *Systems Engineering and Analysis: Fourth Edition*. Pearson College Div., 2006.

[6] SEBoK Editorial Board. Guide to the Systems Engineering Body of Knowledge (SEBoK), 2025.

[7] Bruce Powel Douglass. *Agile systems engineering*. 2016.

[8] William M Kimmel, Patricia M Beauchamp, Margaret A Frerking, Teresa R Kline, Kenny Koorosh Vassigh, Douglas E Willard, Michael A Johnson, and Timothy G Trenkle. Technology Readiness Assessment Best Practices Guide. 2020.

[9] Marc F. Austin and Donald M. York. System Readiness Assessment (SRA) an illustrative example. *Procedia Computer Science*, 44(C):486–496, 2015.

[10] Justin Garrison and Kris Nova. *Cloud Native Infrastructure : Patterns for Scalable Infrastructure and Applications in a Dynamic Environment*. O'Reilly, Beijing, [China, first edit edition, 2018.

[11] Thomas Boillat and Christine Legner. Why do companies migrate towards cloud enterprise systems? A post-implementation perspective. *Proceedings - 16th IEEE Conference on Business Informatics, CBI 2014*, 1:102–109, 2014.

[12] Flexera. 2023 State of the Cloud Report. Technical report, Flexera, 2023.

[13] Flexera. 2024 State of the Cloud Report. Technical report, 2024.

[14] Flexera. 2025 State of the Cloud Report. Technical report, 2025.

[15] Thomas M Booth and Sudipto Ghosh. Machine learning model cards toward model-based system engineering analysis of resource-limited systems. *Signal Processing, Sensor/Information Fusion, and Target Recognition XXXII*, 12547, 2023.

[16] Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206(1):79–111, 2014.

[17] Jianmei Guo, Krzysztof Czarnecki, Sven Apel, Norbert Siegmund, and Andrzej Wasowski. Variability-aware performance prediction: A statistical learning approach. *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013 - Proceedings*, pages 301–311, 2013.

[18] Kenneth Hoste, Aashish Phansalkar, and Lieven Eeckhout. Performance Prediction based on Inherent Program Similarity. *2006 International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 114–122, 2006.

[19] Pavel Valov, Jean Christophe Petkovich, Jianmei Guo, Sebastian Fischmeister, and Krzysztof Czarnecki. Transferring performance prediction models across different hardware platforms. *ICPE 2017 - Proceedings of the 2017 ACM/SPEC International Conference on Performance Engineering*, (2):39–50, 2017.

[20] Gartner. Use Continuous Modernization to Optimize Legacy Applications. Technical report, 2024.

[21] Capt Gary Jones, Edward White, Lt Col, Erin T Ryan, Lt Col, and Jonathan D Ritschel. Operating and Support Costs Weapon Systems. *Defense ARJ*, 21(1):442–464, 2014.

[22] Gregory C. Allen and Isaac Golston. Updating Augustine's Law: Fighter Aircraft Cost Growth in the Age of AI and Autonomy, 2024.

[23] J. A. Estefan. Survey of Model-Based Systems Engineering (MBSE) Methodologies. *INCOSE MBSE Initiative*, 25(8):1–12, 2008.

[24] Forrester. Mathematical Optimization And Machine Learning: Your Perfect AI Tech Team. *Forrester Opportunity Snapshot: a Custom Study Commissioned By Gurobi*, (March):1–12, 2020.

[25] Thomas M Booth. A MODULAR SIMULATION-BASED MBSE APPROACH APPLIED TO A CLOUD-BASED SYSTEM. *INCOSE International Symposium*, 2024.

[26] Thomas M. Booth and Sudipto Ghosh. A Gradient Descent Multi-Algorithm Grid Search Optimization of Deep Learning for Sensor Fusion. *SysCon 2023 - 17th Annual IEEE International Systems Conference, Proceedings*, pages 1–8, 2023.

[27] Alexander Böhm, Jens Dittrich, Niloy Mukherjee, Ippokratis Pandis, and Rajkumar Sen. Operational analytics data management systems. *Proceedings of the VLDB Endowment*, 9(13):1601–1604, 2015.

[28] Massimo Pezzini, Donald Feinberg, Nigel Rayner, and Roxane Edjlali. Hybrid Transaction/-Analytical Processing Will Foster Opportunities for Dramatic Business Innovation. Technical report, Gartner, 2014.

[29] United States Department of Defense. DoD Data Strategy. Technical report, 2020.

[30] United States Department of Defense. Data, Analytics, and Artificial Intelligence Adoption Strategy. Technical report, 2023.

[31] Zhamak Dehghani. *Data Mesh*. 2022.

[32] Jeff A. Estefan and Tim Weilkiens. MBSE methodologies. In *Handbook of Model-Based Systems Engineering*, pages 47–85. 2023.

[33] Robert Karban, Nerijus Jankevičius, and Maged Elaasar. ESEM: Automated Systems Analysis using Executable SysML Modeling Patterns. *INCOSE International Symposium*, 26(1):1–24, 2016.

[34] Sanford A. Friedenthal. *A Practical Guide to SysML*. Elsevier, 2015.

[35] John M Borky and Thomas H Bradley. *Effective Model-Based Systems Engineering*. Springer, 1 edition, 2019.

[36] Jerome Hugues and Joe Yankel. From Model-Based Systems and Software Engineering to ModDevOps, 2021.

[37] Giovanni Di Orio, José Barata, Carlos Sousa, and Luis Flores. Control system software design methodology for automotive industry. *Proceedings - 2013 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2013*, (Lld):3848–3853, 2013.

[38] Scott Ranville. Practical application of model-based software design for automotive. *SAE Technical Papers*, 111:490–494, 2002.

[39] Julie Cohen, Tom Merendino, and Rob Wojcik. Modeling to Support DoD Acquisition Lifecycle Events (Version 1.4). Technical Report April, 2022.

[40] Ronald J Torok and Clifford A Whitcomb. *Naval Postgraduate School Monterey, California Thesis Using Model-Based Systems Engineering Methods To Capture a Department of Defense Acquisition Life Cycle*. PhD thesis, NAVAL POSTGRADUATE SCHOOL, 2021.

[41] Michael T Shutlock. *Applying Digital Engineering to Defense Acquisitions through Model-based Systems Engineering and Discrete Event Simulation*. PhD thesis, Air Force Institute of Technology, 2024.

[42] Bruce Powel Douglass. Harmony-SW Modeling Standards for use with MDA , UML , and Rhapsody. 2012.

[43] Bruce Powel Douglass. Harmony MBSE Modeling Standards. pages 0–50, 2015.

[44] Bruce Powel Douglass. Harmony aMBSE Deskbook Version 1.02. 2017.

[45] Orit Hazzan and Yael Dubinsky. The Agile Manifesto. *SpringerBriefs in Computer Science*, 0(9783319101569):9–14, 2001.

[46] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, Robert C. Martin, Steve Mellor, Dave Thomas, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Ken Schwaber, and Jeff Sutherland. The Agile Manifesto, 2001.

[47] Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles. A survey of DevOps concepts and challenges. *ACM Computing Surveys*, 52(6), 2019.

[48] Christof Ebert and Frank Kirschke-Biller. Agile Systems Engineering. *IEEE Software*, 38(4):7–15, 2021.

[49] FinOps Foundation. FinOps Organization, 2023.

[50] Damien Hardy, Marios Kleanthous, Isidoros Sideris, Ali G. Saidi, Emre Ozer, and Yiannakis Sazeides. An analytical framework for estimating TCO and exploring data center design space. *ISPASS 2013 - IEEE International Symposium on Performance Analysis of Systems and Software*, pages 54–63, 2013.

[51] Daniel Fleischer. WHITE P APER Forecasting Total Cost of Ownership for Initial Deployments of Server Blades. 2006.

[52] Jonathan Koomey, K Brill, Pitt Turner, John Stanley, and Bruce Taylor. A Simple Model for Determining True Total Cost of Ownership for Data Centers. *Uptime Institute, Llc*, 2.1(March):1–9, 2008.

[53] Jg Koomey, C Belady, and M Patterson. Assessing trends over time in performance, costs, and energy use for servers. *Lawrence Berkeley National Laboratory, Stanford University, Microsoft Corporation, and Intel Corporation, Tech. Rep.*, 2009.

[54] Jannice Felicia Salindeho, Jimmy H. Moedjahedy, and Oktoverano Lengkong. Cost-Benefit Analysis of Cloud Computing in Education Using the Base Cost Estimation Model. *3rd International Conference on Cybernetics and Intelligent Systems, ICORIS 2021*, pages 1–5, 2021.

[55] Hong Linh Truong and Schahram Dustdar. Composable cost estimation and monitoring for computational applications in cloud computing environments. *Procedia Computer Science*, 1(1):2175–2184, 2010.

[56] Kyungwoon Cho and Hyokyung Bahn. A Cost Estimation Model for Cloud Services and Applying to PC Lab Platforms. 2020.

[57] Bhathiya Wickremasinghe, Rodrigo N. Calheiros, and Rajkumar Buyya. CloudAnalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications. *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*, pages 446–452, 2010.

[58] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, and A F De Rose. CloudSim : a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. (August 2010):23–50, 2011.

[59] Saurabh Kumar Garg and Rajkumar Buyya. NetworkCloudSim: Modelling parallel applications in cloud simulations. *Proceedings - 2011 4th IEEE International Conference on Utility and Cloud Computing, UCC 2011*, (Vm):105–113, 2011.

[60] Edward R Carroll and Robert J Malins. Systematic Literature Review: How is Model-Based Systems Engineering Justified? Technical report, Sandia National Laboratories, 2016.

[61] OMG. OMG Systems Modeling Language (SysML) ver 1.6. 2003.

[62] OMG. Semantics of a Foundational Subset for Executable UML Models (fUML). *The Object Management Group*, (October):441, 2021.

[63] Henson Graves, Stephen Guest, Jeff Vermette, Yvonne Bijan, Harold Banks, Greg Whitehead, and Bill Ison. Air Vehicle Model-Based Design and Simulation Pilot. Technical report, 2009.

[64] Pier Davide Ciampa, Björn Nagel, and Gianfranco La Rocca. A mbse approach to mdao systems for the development of complex products. *Aiaa Aviation 2020 Forum*, 1 PartF:1–31, 2020.

[65] Justin S. Gray, John T. Hwang, Joaquim R.R.A. Martins, Kenneth T. Moore, and Bret A. Naylor. OpenMDAO: an open-source framework for multidisciplinary design, analysis, and optimization. *Structural and Multidisciplinary Optimization*, 59(4):1075–1104, 2019.

[66] Santiago Balestrini-robinson, Dane F Freeman, and Daniel C Browne. An object-oriented and executable SysML framework for rapid model development. *Procedia - Procedia Computer Science*, 44:423–432, 2015.

[67] Steven W Mitchell. Efficient Management of Configurations in the Model-Based System Development of a Common Submarine Combat System, 2011.

[68] Michael LaSorda. *Applying Model-Based Systems Engineering To Architecture Optimisation and Selection During System Acquisition*. PhD thesis, Colorado State University, 2018.

[69] Sebastian J.I. Herzig, Sanda Mandutianu, Hongman Kim, Sonia Hernandez, and Travis Imken. Model-transformation-based computational design synthesis for mission architecture optimization. *IEEE Aerospace Conference Proceedings*, 2017.

[70] Amazon Web Services. AWS Cost Optimization, 2024.

[71] Microsoft. Azure Cost Optimization Design Principles, 2023.

[72] Steven J. Simske. *META-ALGORITHMICS*. Wiley, 2013.

[73] Sen He, Glenna Manns, John Saunders, Wei Wang, Lori Pollock, and Mary Lou Soffa. A statistics-based performance testing methodology for cloud applications. *ESEC/FSE 2019 - Proceedings of the 2019 27th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 188–199, 2019.

[74] Clay Mathematics Institute. Unsolved Navier-Stokes Equation, 2024.

[75] J H Ferziger, M Perić, and R L Street. *Computational Methods for Fluid Dynamics*. Springer International Publishing, 2019.

[76] Eric R Pardyjak and Michael J. Brown. Evaluation of a fast response urban wind model-comparison to single building wind-tunnel data. *Los Alamos National Labs*, (February 2001), 2001.

[77] Michael D. Williams, Michael J. Brown, David Boswell, Balwinder Singh, and Eric Pardyjak. Testing of the quic-plume model with wind-tunnel measurements for a high-rise building. *13th Joint Conference on the Applications of Air Pollution Meteorology with the Air and Waste Management Association*, (January):267–276, 2004.

[78] Balwinder Singh, Bradley S. Hansen, Michael J. Brown, and Eric R. Pardyjak. Evaluation of the QUIC-URB fast response urban wind model for a cubical building array and wide building street canyon. *Environmental Fluid Mechanics*, 8(4):281–312, 2008.

[79] Thomas Booth and Eric Pardyjak. Validation of a data assimilation technique for an urban wind model. *Symposium on the Urban Environment*, 5(3), 2006.

[80] Thomas Michael Booth. Validation of a Data Assimilation Technique for an Urban Wind Model. (May), 2012.

[81] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. M5 accuracy competition: Results, findings, and conclusions. *International Journal of Forecasting*, 38(4):1346–1364, 2022.

[82] Boris N. Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. In *International Conference on Learning Representations*, 2020.

[83] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3):1181–1191, 2020.

[84] Bryan Lim, Sercan Arık, Nicolas Loeff, and Tomas Pfister. Temporal Fusion Transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, 37(4):1748–1764, 2021.

[85] Hilde J. P. Weerts, Andreas C. Mueller, and Joaquin Vanschoren. Importance of Tuning Hyperparameters of Machine Learning Algorithms. *arXiv*, 2020.

[86] J. Bergstra, D. Yamins, and D. D. Cox. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. *Proceedings of the 30th International Conference on Machine Learning*, 28, 2013.

[87] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[88] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[89] Shaun M Howard and Michael S Lewicki. *Deep Learning for Sensor Fusion*. PhD thesis, Case Western Reserve University, 2017.

[90] Shuochao Yao, Shaohan Hu, Yiran Zhao, Aston Zhang, and Tarek Abdelzaher. DeepSense: A unified deep learning framework for time-series mobile sensing data processing. *26th International World Wide Web Conference, WWW 2017*, pages 351–360, 2017.

[91] John Richards, David Piorkowski, Michael Hind, Stephanie Houde, and Aleksandra Mojsilović. A Methodology for Creating AI FactSheets. 2020.

[92] Emily M Bender. Data Statements for Natural Language Processing : Toward Mitigating System Bias and Enabling Better Science. 6:587–604, 2018.

[93] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé Iii, and Kate Crawford. Datasheets for datasets. *Communications of the ACM*, 64(12):86–92, 2021.

[94] Kasia S. Chmielinski, Sarah Newman, Matt Taylor, Josh Joseph, Kemi Thomas, Jessica Yurkofsky, and Yue Chelsea Qiu. The Dataset Nutrition Label (2nd Gen): Leveraging Context to Mitigate Harms in Artificial Intelligence. 2022.

[95] Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. Model cards for model reporting. *FAT* 2019 - Proceedings of the 2019 Conference on Fairness, Accountability, and Transparency*, pages 220–229, 2019.

[96] M. Arnold, D. Piorkowski, D. Reimer, J. Richards, J. Tsay, K. R. Varshney, R. K.E. Bellamy, M. Hind, S. Houde, S. Mehta, A. Mojsilovic, R. Nair, K. Natesan Ramamurthy, and A. Olteanu. FactSheets: Increasing trust in AI services through supplier's declarations of conformity. *IBM Journal of Research and Development*, 63(4-5), 2019.

[97] Erik Blasch, James Sung, and Tao Nguyen. Multisource AI Scorecard Table for System Evaluation. *Association for the Advancement of Artificial Intelligence (AAAI)*, 2021.

[98] Kaitlin Henderson and Alejandro Salado. Value and benefits of model-based systems engineering (MBSE): Evidence from the literature. *Systems Engineering*, 24(1):51–66, 1 2021.

[99] Tim Weilkiens. Adoption of MBSE in an Organization. In *Handbook of Model-Based Systems Engineering*, pages 1–19. Springer Nature Switzerland, 2022.

[100] Bernard P. Zeigler, Saurabh Mittal, and Mamadou Kaba Traore. MBSE with/out simulation: State of the art and way forward. *Systems*, 6(4):1–18, 2018.

[101] Robert Nshimirimana, Ajith Abraham, and Gawie Nothnagel. *A multi-objective particle swarm for constraint and unconstrained problems*, volume 33. Springer London, 2021.

[102] Davide Fioriti, Giovanni Lutzemberger, Davide Poli, Pablo Duenas-Martinez, and Andrea Micangeli. Coupling economic multi-objective optimization and multiple design options: A business-oriented approach to size an off-grid hybrid microgrid. *International Journal of Electrical Power and Energy Systems*, 127(December 2020):106686, 2021.

[103] Jessica Ryan, Shahram Sarkani, and Thomas Mazzuchi. Leveraging Variability Modeling Techniques for Architecture Trade Studies and Analysis. *Systems Engineering*, 17(1):10–25, 2014.

[104] Mbugua Samuel Thaiya, Korongo Julia, and Samuel Mbugua. On Software Modular Architecture: Concepts, Metrics and Trends. *International Journal of Computer & Organization Trends*, 12(1):3–10, 2022.

[105] R. Honcak. A MBSE approach for Virtual Verification & Validation of E-Drives with Digital Twins. *Dritev 2024*, pages 159–174, 2024.

[106] Imane Bouhali, Vincent Idasiak, Jacques Martinez, Faida Mhenni, Jean Yves Choley, Luca Palladino, and Frederic Kratz. A Collaboration Framework Using Digital Twin for Dynamic Simulation and Requirements Verification Based on MBSE and the MIC Concept. *SysCon 2024 - 18th Annual IEEE International Systems Conference, Proceedings*, pages 1–8, 2024.

[107] John J. Quartuccio and Kristin M. Giammarco. A Model-Based Approach to Investigate Emergent Behaviors in Systems of Systems. In *Engineering Emergence. A Modeling and Simulation Approach*, page 70. CRC Press, 2018.

[108] Mary Ann Cummings. Identifying and Quantifying Emergent Behavior Through System of Systems Modeling and Simulation. 2015.

[109] M L Griss. Software reuse: From library to factory. *IBM Systems Journal*, 32(4):548–566, 1993.

[110] Oladele Junior Adeyeye and Ibrahim Akanbi. Optimization in Systems Engineering: a Review of How Data Analytics and Optimization Algorithms Are Applied. *Computer Science & IT Research Journal*, 5(4):809–823, 2024.

[111] Claude E. Shannon. A Mathematical Theory of Communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[112] Joaquim R.R.A. Martins and Andrew King. *Engineering Design Optimization*. 2021.

[113] Rudolf M. Smaling and Olivier L. De Weck. Fuzzy pareto frontiers in multidisciplinary system architecture analysis. *Collection of Technical Papers - 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 5(September):2789–2806, 2004.

[114] Christopher A. Mattson and Achille Messac. Concept selection using s-pareto frontiers. *AIAA Journal*, 41(6):1190–1198, 2003.

[115] Brian Sauser, Jose E. Ramirez-Marquez, Romulo Magnaye, and Weiping Tan. A Systems Approach to Expanding the Technology Readiness Level within Defense Acquisition. 2009.

[116] Patrick K. Malone. Applying system readiness levels to cost estimates - A case study. *IEEE Aerospace Conference Proceedings*, 2018-March:1–18, 2018.

[117] DUSD(S&T). Technology Readiness Assessment (TRA) Deskbook. Technical Report September, 2003.

[118] Brian Sauser, Ryan Gove, Eric Forbes, and Jose Emmanuel Ramirez-Marquez. Integration maturity metrics: Development of an integration readiness level. *Information Knowledge Systems Management*, 9(1):17–46, 2010.

[119] Jennifer M. Long. Integration readiness levels. *IEEE Aerospace Conference Proceedings*, pages 1–9, 2011.

[120] Mary Sánchez-Gordón and Ricardo Colomo-Palacios. Security as Culture: A Systematic Literature Review of DevSecOps. *Proceedings - 2020 IEEE/ACM 42nd International Conference on Software Engineering Workshops, ICSEW 2020*, pages 266–269, 2020.

[121] Alexandru Iosup, Simon Ostermann, Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Transactions on Parallel and Distributed Systems*, 22(6):931–945, 2011.

[122] Steven C Chapra and Raymond P Canale. *Numerical methods for engineers : with software and programming applications*. McGraw-Hill, Boston, 4th ed. edition.

[123] Dominik Kreuzberger, Niklas Kuhl, and Sebastian Hirschl. Machine Learning Operations (MLOps): Overview, Definition, and Architecture. *IEEE Access*, 11(February):31866–31879, 2023.

[124] R. van der Meulen and T. McCall. Gartner Says Nearly Half of CIOs Are Planning to Deploy Artificial Intelligence., 2018.

[125] Franklin E. White. Data Fusion Lexicon. *The DATA FUSION SUBPANEL of the Joint Directors of Laboratories, Technical Panel for C3*, 15(0704), 1991.

[126] Belur V. Dasarathy. Sensor fusion potential exploitation-innovative architectures and illustrative applications. *Proceedings of the IEEE*, 85(1):24–38, 1997.

[127] Boris R. Bracio, Wolfgang Horn, and Dietmar P.F. Moeller. Sensor fusion in biomedical systems. *Annual International Conference of the IEEE Engineering in Medicine and Biology - Proceedings*, 3(C):1387–1390, 1997.

[128] Alan N. Steinberg, Christopher L. Bowman, and Franklin E. White. Revisions to the JDL data fusion model. *Sensor Fusion: Architectures, Algorithms, and Applications III*, 3719(March 1999):430, 1999.

[129] Ramon F. Brena, Antonio A. Aguileta, Luis A. Trejo, Erik Molino-Minero-Re, and Oscar Mayora. Choosing the best sensor fusion method: A machine-learning approach. *Sensors (Switzerland)*, 20(8):1–22, 2020.

[130] Bahador Khaleghi, Alaa Khamis, Fakhreddine O. Karray, and Saiedeh N. Razavi. Multisensor data fusion: A review of the state-of-the-art. *Information Fusion*, 14(1):28–44, 2013.

[131] Ivan Miguel Pires, Nuno M. Garcia, Nuno Pombo, and Francisco Flórez-Revuelta. From data acquisition to data fusion: A comprehensive review and a roadmap for the identification of activities of daily living using mobile devices. *Sensors (Switzerland)*, 16(2), 2016.

[132] NASA. DASH Link, 2018.

[133] Dan Lovell and Hussain Sultan. Augmenting Trino with DuckDB's AsOf Join and PyTorch, 2025.

[134] Shuo Liu, Huan Liu, Vijay John, Zheng Liu, and Erik Blasch. Enhanced situation awareness through CNN-based deep multimodal image fusion. *Optical Engineering*, 59(05):1, 2020.

[135] Jeffrey J. Danielson and Dean B. Gesch. Global Multi-resolution Terrain Elevation Data 2010 (GMTED2010). *U.S. Geological Survey Open-File Report 2011-1073*, 2010:26, 2011.

[136] FAA. 14 CFR, Chapter I, Subchapter C, Part 43 - Maintenance, Preventive Maintenance, Rebuilding, and Alteration, 2025.

[137] Nitish Shirish Keskar, Jorge Nocedal, Ping Tak Peter Tang, Dheevatsa Mudigere, and Mikhail Smelyanskiy. On large-batch training for deep learning: Generalization gap and sharp minima. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, pages 1–16, 2017.

[138] Manas Bajaj, Sanford Friedenthal, and Ed Seidewitz. Systems Modeling Support for Digital Engineering. *INCOSE Insight*, pages 19–24, 2022.

[139] Data Device Corporation. Designer's Guide to MIL-STD-1553. Technical report, 2012.

[140] Ralph D'Agostino and E. S. Pearson. Tests for Departure from Normality. *Biometrika Trust*, 60(3):613–622, 1973.

Appendix A
RNN Model Card Extension YAML Example

```yaml
---
rnn-based_model_card:
  name: Base+removeVar
  RNN_algorithm: LSTM
  model_metrics:
    RNN_properties: {epochs: 3500, hidden_units: 20, learning_rate: 0.0007, n_inputs: 23,
      n_outputs: 1, sequence_length: 16}
    input_data_frequency: 16
    input_variable_names: [time, RALT, PSA, PI, PT, ALTR, IVV, VSPS, FPAC, BLAC, CTAC,
      TAS, CAS, GS, CASS, WS, PTCH, ROLL, DA, TAT, SAT, LATP, LONP]
    model_datasheet:
      testing_files: [687200107241524.parquet, 687200104301119.parquet,
        687200104261527.parquet, 687200104181334.parquet, 687200107301239.parquet,
        687200104170717.parquet, 687200107170234.parquet]
      testing_files_location: /mnt/SPIE/ml-model-cards/data
      training_files: [687200107101600.parquet, 687200107060930.parquet,
        687200107192334.parquet, 687200107150546.parquet, 687200107311025.parquet,
        687200107181544.parquet, 687200104162039.parquet, 687200104202027.parquet,
        687200107251652.parquet, 687200104181127.parquet, 687200107251002.parquet,
        687200107122323.parquet, 687200107171131.parquet]

      training_files_location: /mnt/SPIE/ml-model-cards/data
      validation_files: [687200107101623.parquet, 687200107060931.parquet,
        687200107181541.parquet, 687200104162032.parquet, 687200104202023.parquet,
        687200107192330.parquet, 687200107150549.parquet]
      validation_files_location: /mnt/SPIE/ml-model-cards/data
    output_variable_names: [ALT]
    performance:
      test: {MSE: 389.1024435718944, MSE_norm: 1.0885067922572489e-06,
        mean_error: 8.00731361488409, min_error: 0.0, std_error: 29.48358127688793,
        max_error: 421.5643005371094, units: ft}
      train: {MSE: 15.48635814304236, MSE_norm: 3.140855255878705e-07,
        mean_error: 13.48635814304236, min_error: 0.00018310546875,
        std_error: 14.29056638518885, max_error: 89.2213134765625, units: ft}
      validation: {MSE: 386.1021435718944, MSE_norm: 4.5026759064453724e-07,
        mean_error: 13.48635814304236, min_error: 0.00018310546875,
        std_error: 14.29056638518885, max_error: 160.2213134765625, units: ft}
  bagging_metrics:
    model_datasheet: {dataset: null}
    n_bootstrap_loops: 3
    performance:
      testing: {max_MSE: null, mean_MSE: 2.933555492745654e-06, min_MSE: null,
        std_MSE: null}
      training: {max_MSE: null, mean_MSE: 1e-07, min_MSE: null, std_MSE: null}
      validation: {max_MSE: null, mean_MSE: null, min_MSE: null, std_MSE: null}
    properties: {N_bootstrap_datasets: 3, replacement: false}
  benchmark:
    dataset: {location: /mnt/SPIE/ml-model-cards/data, name: example_flight.parquet}
    device1:
      device_ID: 1
      name: CovidPC
      RAM:
        CAS_latency: 16
        first_word_latency: {speed: 8.889, units: ns}
        size: 32
        speed: 3600
        units: GB
        voltage: {units: V, value: 1.35}
      processor:
        L1Cache_Data: {units: kB, value: 192}
        L1Cache_Instruction: {units: kB, value: 192}
        L2Cache: {units: kB, value: 3072}
        L3Cache: {units: MB, value: 32}
        TDP: {units: W, value: 65}
        core_clock: {units: GHz, value: 3.6}
        name: AMD Ryzen 5 3600
        ncores: 6
```

```
        type: CPU
    performance:
      data_processing_latency: {units: seconds, value: 0.2591249942779541}
      model_inference_latency: {units: seconds, value: 0.401350736618042}
      nonvolatile_memory_used: {units: MB, value: 7.4619035720825195}
      throughput: {units: Kbps, value: 2.8}
      validation: {MSE: 386.1021435718944, MSE_norm: 4.5026759064453724e-07,
        mean_error: 13.48635814304236, min_error: 0.00018310546875,
        std_error: 14.29056638518885, max_error: 160.2213134765625, units: ft}
      volatile_memory_used: {units: MB, value: 2084.9192708333335}"
...
```

Appendix B
SysML Model Card YAML Example

```yaml
---
sysml_model_card :
  name : 'NASA Sensor System SysML model1'
  file : 'simple_nasa_avionics_sensor_model.mdzip'
  file_location : '/mnt/SPIE/sysml-model-cards-for-dl-sensor-fusion'
  sysml_version : '1.6'
  data_model :
    data :
      ALT  : {size : 16, rate : 4}
      time : {size : 32, rate : 1}
      FPAC : {size : 32, rate : 16}
      BLAC : {size : 32, rate : 16}
      CTAC : {size : 32, rate : 16}
      VRTG : {size : 32, rate : 8}
      LATG : {size : 32, rate : 4}
      LONG : {size : 32, rate : 4}
      RALT : {size : 16, rate : 8}
      ALTR : {size : 16, rate : 4}
      IVV  : {size : 16, rate : 16}
      VSPS : {size : 16, rate : 1}
      PSA  : {size : 16, rate : 2}
      PI   : {size : 16, rate : 2}
      PT   : {size : 16, rate : 2}
      TAS  : {size : 16, rate : 4}
      CAS  : {size : 16, rate : 4}
      GS   : {size : 16, rate : 4}
      WS   : {size : 16, rate : 4}
      CASS : {size : 16, rate : 1}
      PTCH : {size : 16, rate : 8}
      ROLL : {size : 16, rate : 8}
      DA   : {size : 16, rate : 4}
      TAT  : {size : 8,  rate : 1}
      SAT  : {size : 8,  rate : 1}
      LATP : {size : 32, rate : 1}
      LONP : {size : 32, rate : 1}
    networks :
      BUS1 :
        constraints :
          throughput : {datarate : 1, units : 'Mbps'}
        data : [RALT, ALTR, IVV, VSPS, PSA, PI, PT, TAS, CAS, GS, WS, CASS, PTCH, ROLL,
          DA, TAT, SAT, ALT]
      BUS2 :
        constraints :
          throughput : {datarate : 1, units : 'Mbps'}
        data : [time, FPAC, BLAC, CTAC, VRTG, LATG, LONG, VSPS, CASS, LATP, LONP, ALT]
...
```