34th Annual INCOSE international symposium
hybrid event
Dublin, Ireland
July 2 - 6, 2024

A MODULAR SIMULATION-BASED MBSE APPROACH APPLIED TO A CLOUD-BASED SYSTEM

Thomas M. Booth
Colorado State University
200 W. Lake Street
Fort Collins, CO 80523-6029
tom.booth@colostate.edu

Sudipto Ghosh
Colorado State University
200 W. Lake Street
Fort Collins, CO 80523-6029
sudipto.ghosh@colostate.edu

Abstract. The current state of the practice of Model-Based Systems Engineering (MBSE) methodologies, system specification models, and executable system simulations lack the capability of simulation execution within a single system specification model with the fidelity required for time-dependent simulation towards the design, analysis, and optimization of complex software-based systems. In this paper we develop and demonstrate a modular simulation-based MBSE approach capable of filling this gap. To accomplish this within a single system specification model we embed a combination of modeled simulation activities, opaque behaviors, and simulation specific functions to execute and manage the time-dependent simulation variable arrays. We applied our MBSE approach on the design and analysis of a hybrid on-prem/cloud data system example that meets the complex, software-based, and time-dependent requirements this approach was built to solve. The results show that our approach produces a modular and time-dependent simulation-enabled system specification model that accurately estimates cloud-based system performance and storage cost as a function of time. Emergent system behaviors observed from the simulation results indicate the system model provides foundational design and analysis capabilities that are required for applying system optimization algorithms.

Keywords. cloud-based system design, model-based systems engineering, time-base simulation, modular design modeling approach

Introduction

Many prominent Model-Based Systems Engineering (MBSE) trade study methodologies are developed with a specification model and at least one simulation model which are usually accomplished with multiple integrated tool sets (e.g. Cameo, MATLAB, Model Center, Excel, etc.) to perform architecture or system optimizations (Herzig et al., 2017; LaSorda et al., 2018; Ryan et al., 2014). In these instances, the system simulation occurs outside of the core MBSE system specification model.

Whereas, our MBSE approach is specifically built to perform the necessary time-dependent simulations inside the core system specification model and tool. The purpose of this is to reduce model and tool integration needs and additionally verify the system specifications through simulation. The goal of our approach is to provide system engineers the capability to build simulation-enabled specification models towards the design, analysis, and optimization of complex, software-based, and time-dependent systems. While our approach was developed to enable system optimization techniques as a core feature, optimization methods and techniques are beyond the scope of this paper. Our use-case based and modular MBSE approach produces sets of simulation-ready system activities that enable system engineers to easily scale the size of the system model to one that is larger or more complex. The modularity and scalability of our approach is derived from the layered Service-Oriented Architecture (SOA) (Broy, 2003) concept that defines loosely-coupled and modular services in a layered architecture. Our approach uses two layers of visual-based Domain Specific Language (DSL) system specification and simulation elements (i.e. services) and a common programming language based layer of simulation functions and utility services as illustrated in Figure 1. We also define two viewpoints to distinguish between specification and simulation perspectives.

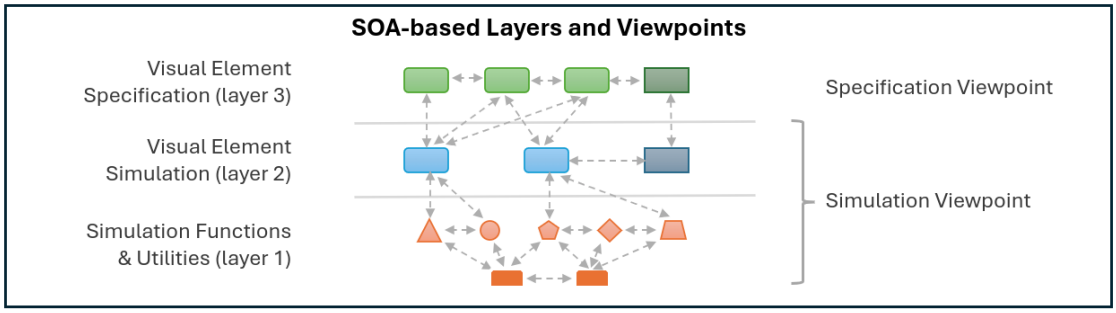

Figure 1. Simulation-enabled MBSE Approach Layers and Viewpoints

The uppermost layer (layer 3) consists of reusable system specification elements that are implemented throughout the model. This reusability is already a best practice in system modeling. This layer represents the specification viewpoint which contains the requirements, logical & physical system architecture, structure, and the behavioral elements that define the system. The next layer down (layer 2) consists of reusable simulation elements that are used by the specification layer (3) to define the simulation configurations and activities. This is also fairly common in MBSE practice. However, one novel aspect of our approach is the application of the lowest layer (layer 1) which consists of simulation functions that perform operations too complex for the visual layer (2) and utilities that initialize and update the time-based arrays. This bottom layer is what allows our approach to produce models with the fidelity needed to capture the state of the system as a function of time. Another novel aspect to our approach is the definition of the last two layers to form the simulation viewpoint which includes the simulation configurations, analysis blocks, simulation activities, opaque behaviors, and simulation functions and utilities.

The layered SOA concept enables scalable and modular systems, therefore since we've based our approach on this architectural concept the models developed are also assumed to be scalable and modular. The MBSE approach developed in this paper is generalized enough to model software-based systems and the constraints specific to their domain. Therefore, we applied it to an exam-

ple cloud-based data system to demonstrate the approach's ability to create a scalable and modular model of a software-based and time-dependent system. Data systems consist of a network of communication channels, applications that transmit data across these channels, and the hardware running these applications or generating the data. Specifically, the example system is a hybrid on-prem/cloud data system that contains an Extract, Transform, Load (ETL) data pipeline for processing large data sets automatically. We demonstrate the behavior of an automated process that transfers data from through an on-prem network which then uploads the data to a cloud-based ETL pipeline which processes the data and saves it to the cloud. This prototype data system is running in Amazon Web Services (AWS) public cloud and provides performance data for the system example.

The rest of this paper discusses previous work related to non-SE cloud Modeling and Simulation (M&S), MBSE simulations, the application of our approach, the demonstration example setup, and the results of the simulations in the Cloud Performance and Cost Modeling, Executable SysML Modeling, Applied MBSE Approach, Demonstration Setup, and Demonstration Results sections, respectively.

Cloud Performance and Cost Modeling

In terms of our cloud-based data system example it is important to note that most modern data systems range from minimal cloud-based storage to full cloud-based storage and compute systems. While the cloud offers near limitless ability to scale up resources on demand to meet its users' need, most organizations don't have the budget to let software developers' applications scale without constraint. According to the 2023 State of the Cloud Report (Flexera, 2023), 66% of the 753 executives and cloud decision-makers surveyed said that cloud usage was "higher than already planned this year" with many saying that they were over budget by 18% for 2022 with an anticipated 30% growth in cloud spend the following year. Additionally, these organizations self estimated they wasted 28% of their cloud budget. Therefore, it is not surprising that the #1 cloud challenge for those surveyed was managing cloud spending. As a result of this immense challenge across organizations, a new discipline called Financial Ops (FinOps) (FinOps Foundation, 2023) has emerged to address the issues of cloud spending. Therefore, a predictive Systems Engineering (SE) approach to cloud system design, analysis, and optimization could save organizations tens of millions of dollars a year when designing, modifying, and maintaining large, complex, cloud-based data systems.

The rest of this section describes the previous work done towards cloud performance and cost M&S with non-SE based methods. For reference, cloud performance and 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 costs that are straight forward calculations. However, cloud costs are difficult to estimate due to the stochastic behavior of software running in shared cloud environments, therefore costs can fluctuate significantly. This is the main reason most cloud cost estimation tools are reactive rather than predictive.

The first of these reactive models is an instant cloud cost estimation tool that was created to estimate Infrastructure as a Service (IaaS) (Cho & Bahn, 2020). This cost estimation model is 99.3% accurate in comparison with actual cloud charges. However, this model deploys a monitoring daemon to a currently running cloud system to measure virtual machine resource usage to estimate cumu-

lative cloud costs. While helpful for instant feedback and monitoring, vs waiting for the monthly bill, this reactive model does not predict resource usage and does not include a system specification model that could inform design and architectural decisions.

The next tool, CloudSim (Calheiros et al., 2011; Wickremasinghe et al., 2010), is an extensible simulation toolkit that enables modeling and simulation of cloud computing system and application provisioning environments. This application has shown to improve the application of Quality of Service (QoS) requirements under fluctuating resources and service demand patterns. However, this toolset requires the CloudSim software environment to be deployed to an on-prem server and the cloud-based software to be written and deployed to the CloudSim environment. While this is a high quality and accurate cloud resource modeling tool and may provide valuable prototype information, the systems engineering must still be accomplished for design, analysis, and optimization capabilities.

Aside from the capabilities of CloudSim, most common software languages provide the capability to model, simulate, and analyze data system performance. However, a visual M&S language, such as executable SysML, provides additional value beyond the capabilities common software languages alone provide. For instance, the Department of Defense has been using SysML system specifications for many value-added and visual-based artifacts for software-based systems such as: Authority To Operate, safety, airworthiness, and nuclear certifications to name a few. Furthermore, the Sandia report (Carroll & Malins, 2016) concludes that there is a significant advantage to project performance when applying an MBSE approach compared to a paper-based SE approach. Therefore, when an MBSE approach includes a simulation-enabled capability, such as ours, it is clear this provides value over using common software language M&S alone.

Executable SysML Modeling

This section reviews other approaches to simulation-based MBSE models for the design of complex and time-dependent systems. The Executable System Engineering Method (ESEM) (Karban et al., 2016) for requirements verification, uses SysML modeling patterns that involve structural, behavioral and parametric diagrams. While capable for requirements verification, their use of parametric diagrams in their demonstration of dynamic power rollups do not work well for time-dependent simulation. In our experience, parametric diagrams are not easily controlled by custom simulation clocks, they execute out of order, and execute multiple times throughout a single time step. This causes time-dependent simulation gradient errors and inaccurate results.

An executable SysML design model for an avionics architecture (Graves et al., 2009) demonstrates that executable SysML design models are capable of simulating data flow on a real-time MIL-STD-1553 aircraft avionics data bus. This simulation includes their own simulation clock to control the fidelity and speed of the simulation. This study integrates the avionics specification model with an exterior simulation model of an existing avionics system. The development effort successfully met the technical challenges to cost effectively build design models with sufficient avionics timing fidelity to provide valid results. This paper reinforces the strength of our postulation that using a visual Domain Specific Language (DSL) coupled with a common programming language to estimate time-dependent system performance leads to better and more efficient designs of complex

data systems. However, their approach relied on integration with an external simulation model and did not appear to be easily scaled to a larger or more complex system model.

Applied MBSE Approach

This section describes the application of our modular simulation-based MBSE approach to an automated hybrid on-prem/cloud ETL pipeline system towards estimation of performance and cloud storage costs as a function of time. The application of this approach is based on a use-case driven Agile SE methodology (Douglass, 2016) that iteratively and incrementally refines the system's structure, requirements, constraints, and behaviors based on stakeholder's requested system behaviors and use cases.

As with any MBSE approach, the tools and languages selected are as important to consider as the approach/methodology itself and should be chosen based on the user's need. We chose SysML 1.6 as the base language to demonstrate our approach because it is currently the most common MBSE language that is also being actively developed. Unfortunately, SysML is not currently defined well enough to sufficiently create a time-dependent and executable model, therefore we use a combination of SysML, fUML, and Groovy for our visual-based DSL and common programming languages, respectively. The Action Language for Foundational UML (ALF) specification could meet some of the requirements for this demonstration, however we found that using this language alone did not meet the simulation capabilities needed to capture the time-dependent simulation variable arrays. We used Cameo Systems Modeler (CSM) Enterprise edition for this demonstration since it is a common tool and supports the modeling languages we chose. However, the choice in tools and languages best suited to apply our MBSE approach may change with the release of the SysML 2.0 specification and tools that support it.

We start out the application of our approach by visually distinguishing between specification and simulation model diagrams for the two viewpoints illustrated in Figure 1. We made this distinction obvious to any users of the model by changing the colors of two simulation elements. We applied a *«Simulation Behavior»* stereotype to the system simulation activity diagrams and the *«Simulation Comment»* stereotype to comments documenting the simulation activity diagrams. The borders of these stereotyped elements were colored blue automatically by CSM to ensure an obvious separation.

Next, the structural ETL data pipeline system specification is built, then the primary ETL use case is modeled by creating system behavior specifications and simulation behavior simultaneously. Later, software performance data measured from a pilot project running in an AWS public cloud is used to provide realistic values for the simulation input variables. Additionally, cloud resource costs use AWS's online pricing information for storage costs. Once the simulation-enable system model is built, we use CSM and Cameo Simulation Toolkit (CST) to execute and record the time-history of the simulation. The Specification Viewpoint and Simulation Viewpoint sections discuss the details of the two perspectives our MBSE approach in the context of this data system example.

Specification Viewpoint

This section describes the system specification aspect of our MBSE approach. Figures 2, 3, and 4 show the internal block diagram, activity diagram, and the block definition diagram of the hybrid on-prem/cloud data system, respectively. These elements and figures are associated with SOA layer 3 of our approach. Figure 2 shows the serial nature of this example system, however the *pipeline* is capable of processing data in parallel. Therefore, this cloud-based *pipeline* is capable of elastically scaling up or down to satisfy the data processing needs which is limited by a variable for maximum number of pipelines. This data processing demand is a function of the amount of data staged in the *Cloud* block's *dataProcessQueueSize* variable which is inherited from the *Data Sub-System* block as shown in 4. The initial use case driving this design is shown in Figure 3 which contains the four call behavior actions defining the behavior of this ETL pipeline. Each of these call behavior actions are activities with well defined system simulation behaviors that can be reused elsewhere as the system expands.

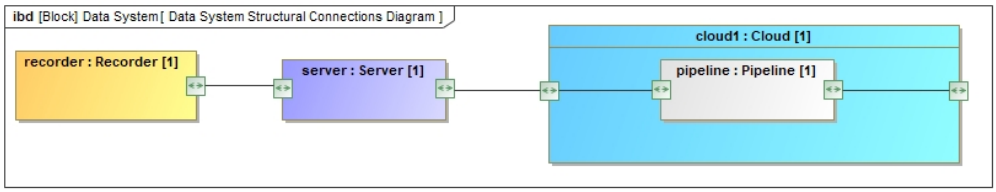

Figure 2. SysML Internal Block Diagram of the ETL Pipeline

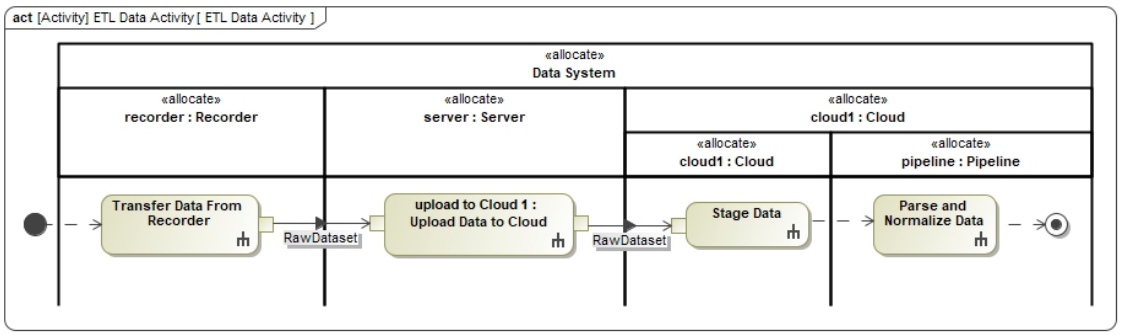

Figure 3. SysML Activity Diagram of the ETL Pipeline

The *Data Sub-System* block in Figure 4 shows a generalized block from which all other block elements in the system model inherit their attributes. The power of inheritance is a key aspect of the system specification and simulation of our MBSE approach. Each sub-system in this data system must have these same value properties in order to make the simulation activities easily reusable. The Simulation Viewpoint section will further describe the role this plays withing the simulation.

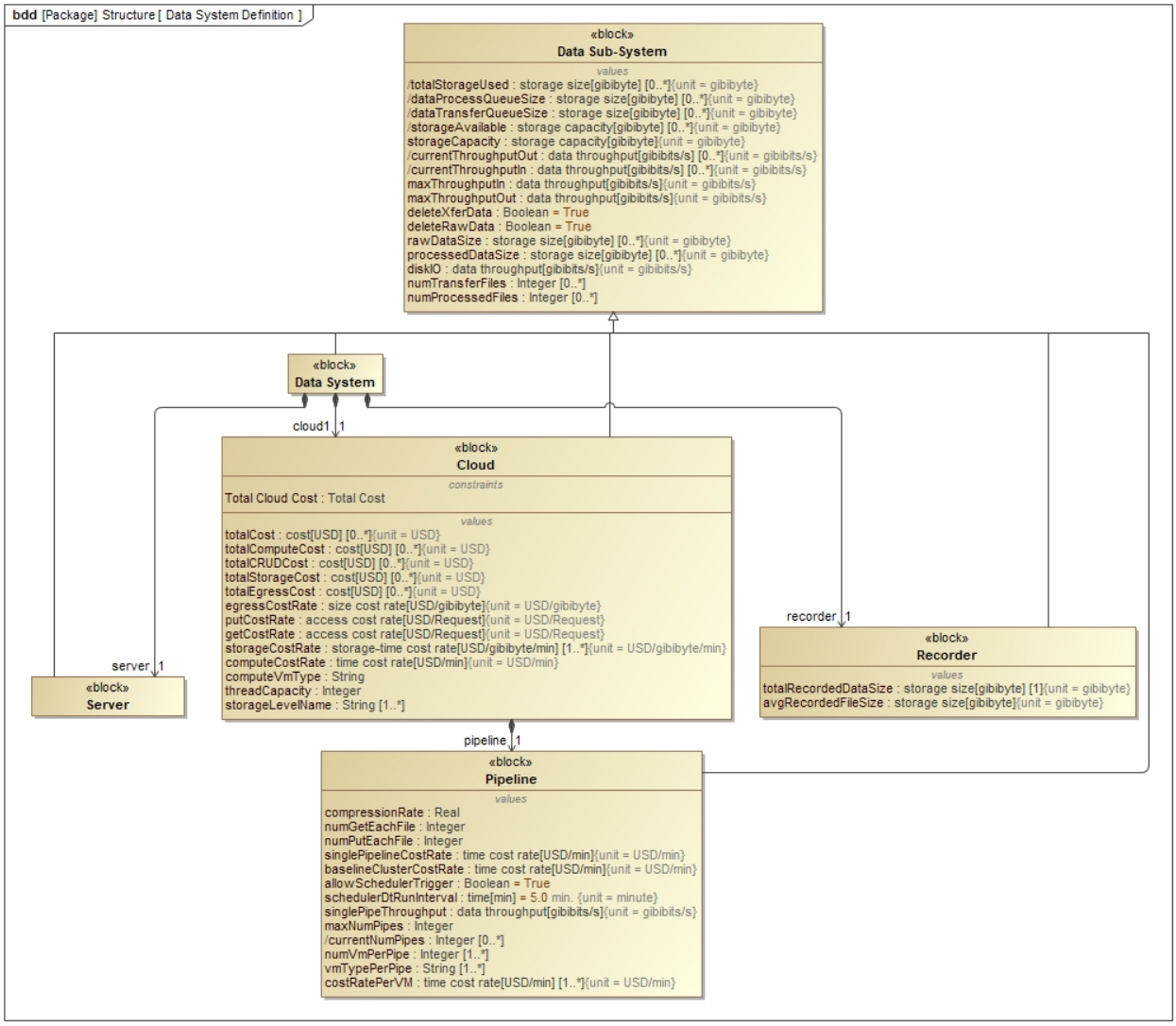

Figure 4. SysML Block Definition Diagram of the ETL Pipeline Composition

Simulation Viewpoint

This section describes the modular and scalable simulation aspect of our approach. When executed, this simulation will provide the state of our system as a function of time based on instantiations of the system specifications. Every simulation activity, opaque behavior, and simulation function is built in such a way to read or modify the time-series variables of the *Data Sub-System* block or any specializations of it. In addition to the power of inheritance, the modularity is further expanded through the use of allocation activity partitions. The activity partitions provide the correct part property context to the simulation activities allocated to them. For instance, Figures 5, 6, and 7 each show reusable call behavior actions allocated to specific parts to provide their simulation context. Of note, these figures also show the application of the *«Simulation Behavior»* and *«Simulation Comment»* stereotypes and coloring scheme to denote that they are simulation viewpoints instead

of specification viewpoints. Additionally, data transfers between sub-systems are done with a generalized *Data* block object for modularity, reuse, and future system model capability upgrades.

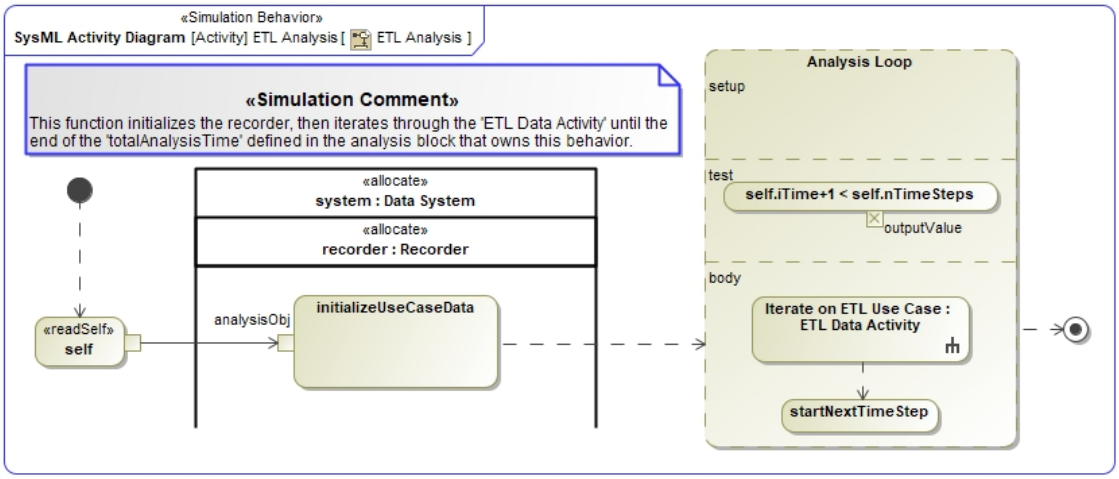

Figure 5. Simulation Activity Diagram of the ETL Analysis Block

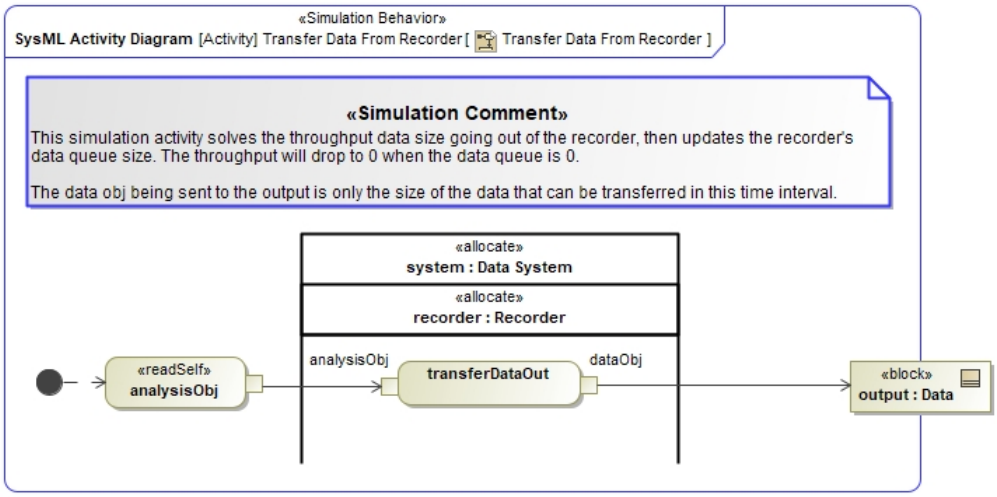

Figure 6. Simulation Activity Diagram of the *Transfer Data From Recorder* Activity

To align with the use-case driven Agile SE methodology, we build a single simulation block for each use case. This helps incrementally build the model and also provides the ability to measure impact to the system for each use-case as it is added. A master simulation block contains all use-case simulation blocks for a full time-synced system simulation. This simulation modularity also makes it easier to troubleshoot the system model if needed.

The use-case simulation block for this example system is called *ETL Analysis* and it defines the simulation variables and constraints used in the execution of this use-case. It is composed of the *Data System* block that defines the system specifications so that it has access to all system specification values. Figure 5 shows the activity diagram of the *ETL Analysis* block which defines our custom simulation clock that is controlled by the length and step size of our simulation time variables. The simulation block contains these timing variables while the custom simulation clock syncs the timing across all sub-system parts.

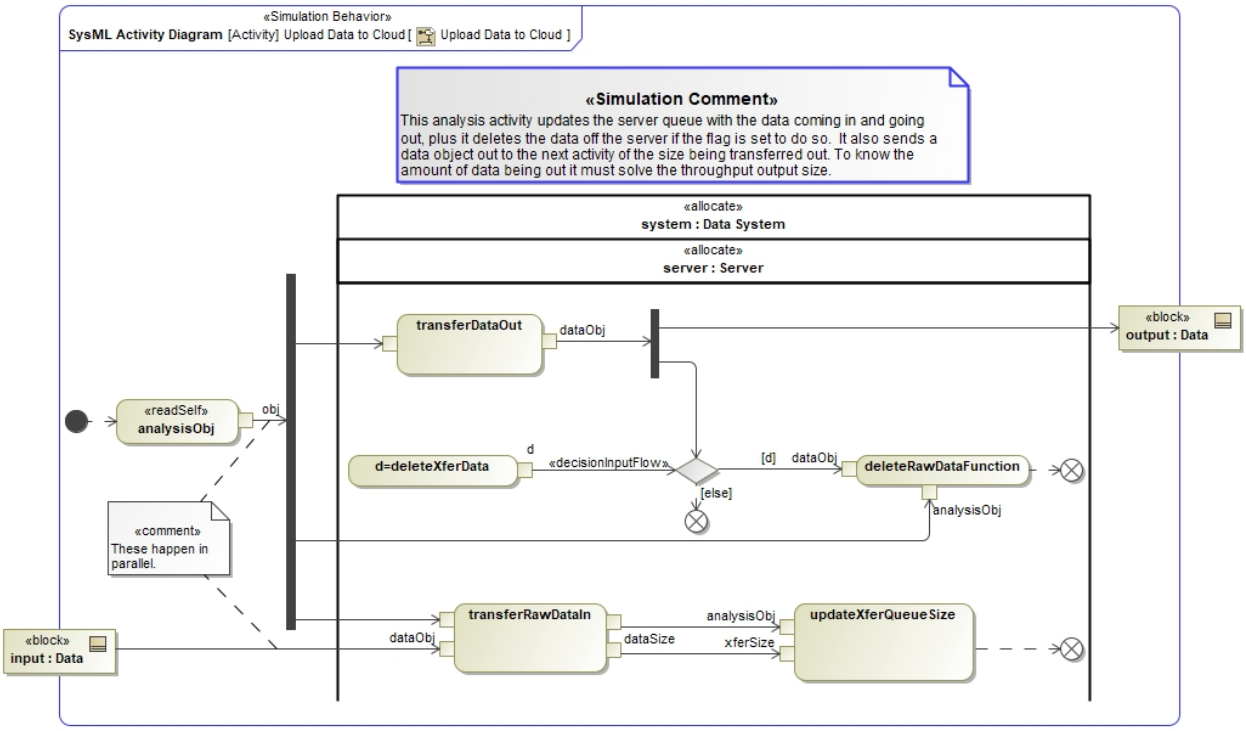

Figure 7. Simulation Activity Diagram of the *Upload Data to Cloud* Activity

Each of the four system specification activities in Figure 3 is defined by a simulation activity to simulate its time dependent behavior. Figures 6 and 7 show two of these simulation activities that map to SOA layer 2 of our approach. The opaque behavior called *transferDataOut* is an example of a reusable simulation activity and is shown in Figures 6 and 7. The specification context of this simulation activity is based on the sub-system its allocated to the activity partition. For instance, *transferDataOut* executes on the *server* sub-system in Figure 7 compared to the *recorder* sub-system in Figure 6. As mentioned previously, this enables system engineers to use these small 'building blocks' to easily scale-up the size or complexity of the model.

Furthermore, the *transferDataOut* simulation activity, or opaque behavior, is defined using Groovy as shown in Figure 8. The body of this opaque behavior makes calls to simulation functions and utilities that are contained in the SOA layer 1 of our approach. Unfortunately, Cameo did not have an easy method to call stand-alone functions so we had to embed these simulation functions into other opaque behaviors and call those functions with Cameo's ALH *callBehavior* API. Two simulation utilities and one simulation function are seen in Figure 8. Specifically, the simulation utilitie *checkListInit* and *updateDataSize* initialize the time-series arrays and updates the specific time-series array for the values provided, respectively. Whereas the simulation functions *solve-LANThroughput* solves throughput of the LAN connection.

Our choice to use Groovy rather than another language was based on simulation execution speed for CST. CST is Java based and therefore Groovy executes the simulation faster than the other languages it supports such as Jython, Jruby, or Beanshell. We believe Groovy is also relatively easy

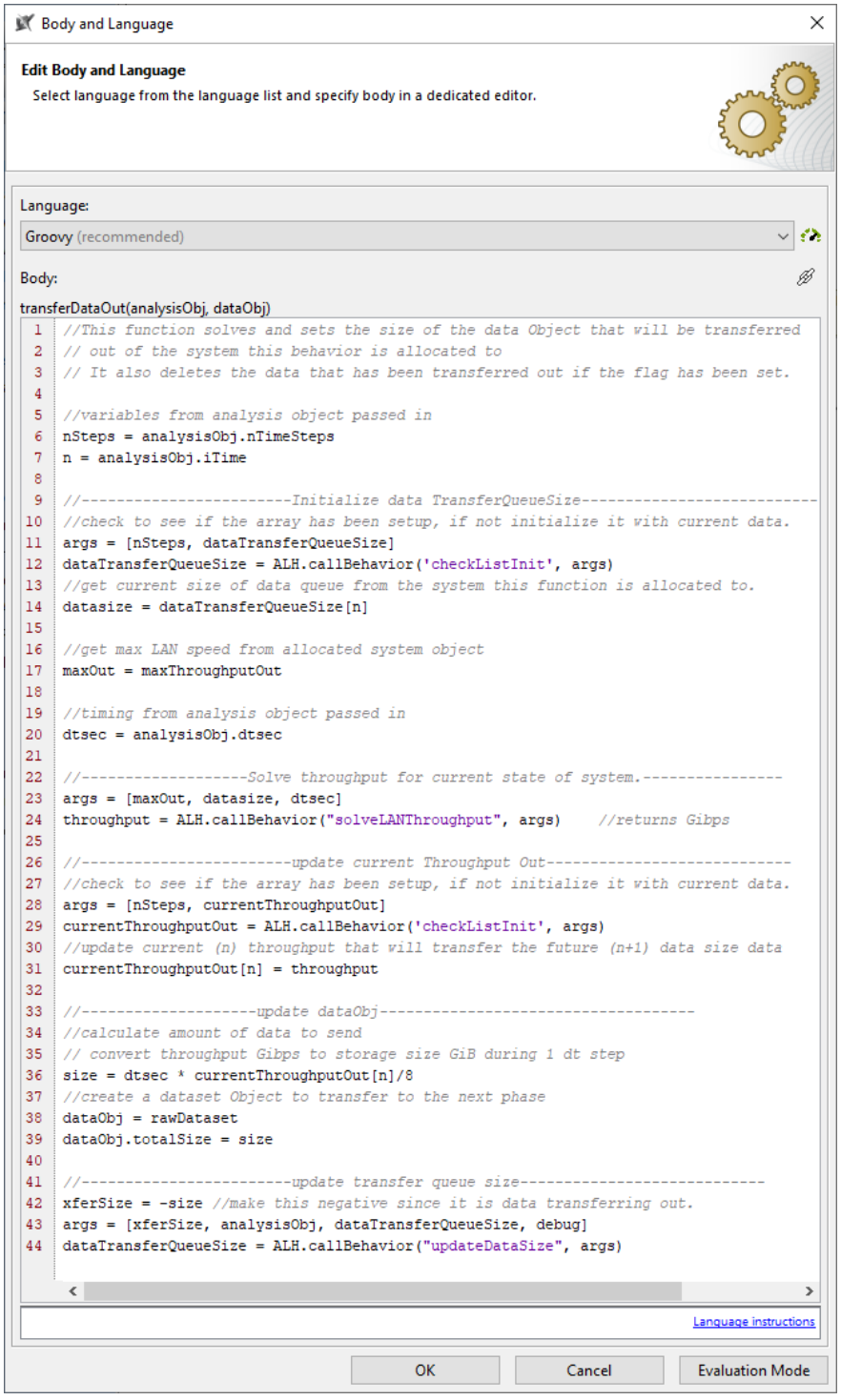

Figure 8. Modular Opaque Behavior Example

to understand for non-software engineers. When this model is integrated with system optimization algorithms in the future the speed of the simulation is be very important, since many simulations will need to be executed to find optimal results.

Demonstration Setup

This section describes the configuration of two simulation systems for our demonstration. Each configuration was built to demonstrate different aspects of four system-specific decision variables that affect: system emergent behaviors, total time to process recorder data, and cloud storage cost as a function of time. Both configurations start with 500 GiB of recorded data, simulation duration of 40 minutes, and time step of 1 minute. The system's data flow simulation is triggered by the presence of data on the recorder (*totalRecorderDataSize*). Table 1 shows the two simulation configurations. To summarize this table and objective of these configurations, scenario 1 has a maximum throughput out (*maxThroughputOut*) of the recorder that is slower than the throughput out of the server, whereas scenario 2 is the opposite. The expected result is that the server in scenario 1 will always transfer data out as fast as it receives it, while the second scenario server will have data build-up in its queue while it is slowly uploaded to the cloud. We varied the two simulation configurations by the four decision variables that we've deemed 'business rules' that affect cloud system performance and cost. The following four business rules describe the function they serve and the potential cost or performance effect.

Table 1. Instantiated ETL Pipeline Scenario Configuration Values

Value Name	Scenario 1	Scenario 2	Units
Simulation Variables			
dtmin	1.0	1.0	min.
totalAnalysisTime	40	40	min.
Specification Variables			
Recorder			
maxThroughputOut	2.6	4.0	Gibps
storageCapacity	1000	1000	GiB
totalRecordedDataSize	500	500	GiB
Server			
deleteXferData	true	false	boolean
maxThroughputIn	4.0	8.0	Gibps
maxThroughputOut	8.0	2.6	Gibps
storageCapacity	400	400	GiB
Cloud			
maxThroughputIn	10	10	Gibps
maxThroughputOut	4.0	4.0	Gibps
storageCapacity	1000	1000	GiB
storageCostRate	5.0E-7	5.0E-7	$/GiB/min.
Pipeline			
deleteRawData	true	false	boolean
singlePipeThroughput	8.66	8.66	Gibps
allowSchedulerTrigger	false	false	boolean
schedulerDtRunInterval	5.0	3.0	min.
maxNumPipes	5	2	integer
compressionRate	0.5	0.5	real

Business rule 1 deletes the server data after it is uploaded to the cloud if *deleteXferData* for the server sub-system is true. You can see from Table 1 that Scenario 1 has this is turned on but scenario 2 does not. This rule does not impact cloud cost or speed, but impacts the size of the server's storage requirements as it is acting like a data storage queue for the cloud.

Business rule 2 is a data pipeline orchestration tool time interval setting. As a general rule of thumb, most commercial pipeline orchestration tools recommend setting an interval to execute the pipeline rather than executing each time data shows up. The variable *schedulerDtRunInterval* sets the pipeline orchestration interval to execute the ETL pipeline when there is data in the cloud processing queue. Scenario 1 is set to run every five minutes while scenario 2 runs every three.

Business rule 3 is a setting to limit the number of parallel pipelines allowed to run at the same time which is set by *maxNumPipes*. Scenario 1 is allowed to run up to 5 pipelines in parallel while scenario 2 can only run up to 2 in parallel. This business rule in conjunction with the interval timing (business rule 2) creates an equivalent possible throughput for the pipeline when taking into account *singlePipeThroughput*. Therefore, Scenario 1 could potentially have 5 parallel pipelines running every 5 minutes, whereas scenario 2 would only have up to 4 pipelines every 6 minutes. Since they both have the same single pipeline throughput it seems reasonable to assume that scenario 1 has the fastest processing time.

Business rule 4 is based on the decision to delete the raw data in the cloud after it has been processed through the data pipeline. This is a cost vs risk decision. If the data pipeline incorrectly processes the data and the raw data has already been deleted then valuable data will be lost. However, keeping both sets of data in the cloud costs money. When *deleteRawData* is true the pipeline will delete this data after it is processed. Scenario 1 deletes data after being processed whereas scenario 2 does not.

All other values for these two scenarios were kept the same. The other values are shown to provide context and are not part of business rules we demonstrate in this paper. Values such as single pipe throughput are tied to the performance specifications of the VMs (AWS EC2 instances) used for the pipeline applications and this can affect performance and cost. The values used in this simulation are from a simple pilot study.

Demonstration Results

This section presents the results from the two simulation scenarios described in the previous section. Figures 9 and 10 are the time-series plots of simulation scenarios 1 and 2, respectively. The lines plotted in these figures are color coded to align to the colors of the sub-systems in the cloud example specification diagram in Figure 2. For instance, the recorder line plot in Figure 9 is light orange to match the orange recorder sub-system part in Figure 2.

Figures 9(a) and 10(a) show the data size in GiB as a function of time for scenario 1 and 2, respectively. These figures display data from the recorder, server, and cloud sub-systems. The *xfer queue* sizes represent the data queued to be transferred to the next sub-system. The storage sizes are the total data being stored on that sub-system while the cloud data sizes represent the amount of the total cloud storage size dedicated to the raw or processed data.

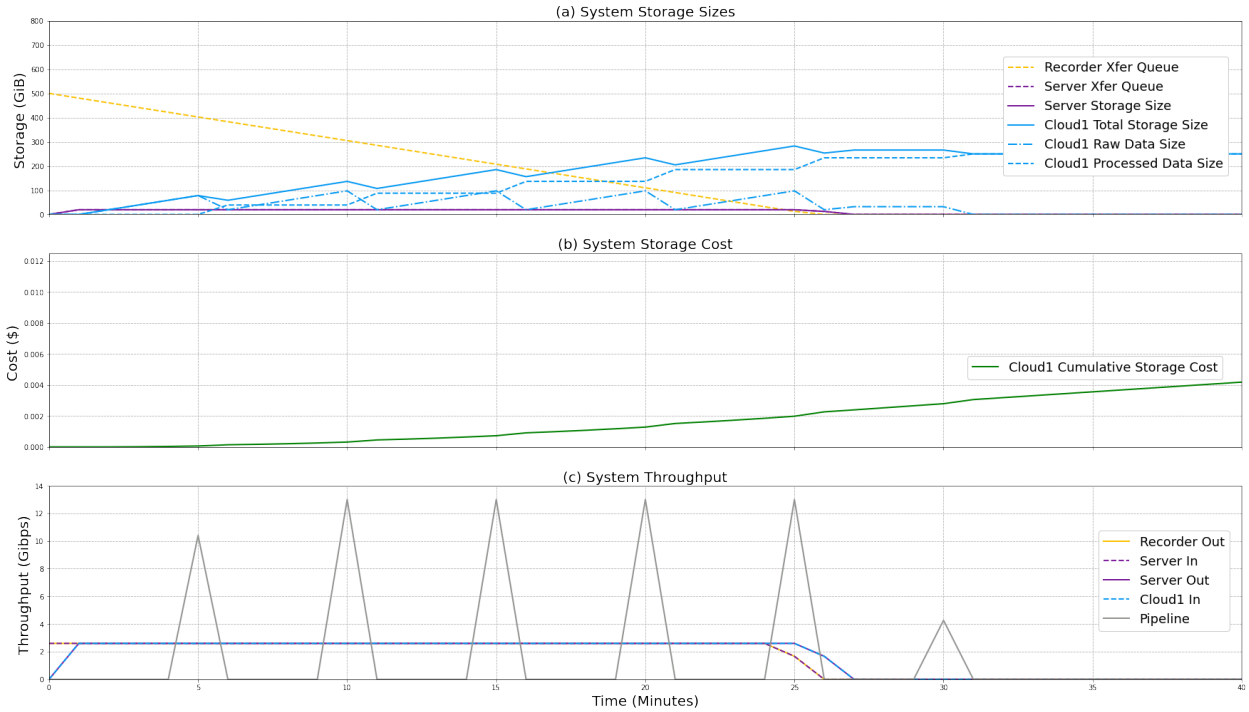

Figure 9. Scenario 1 Performance and Cost Results

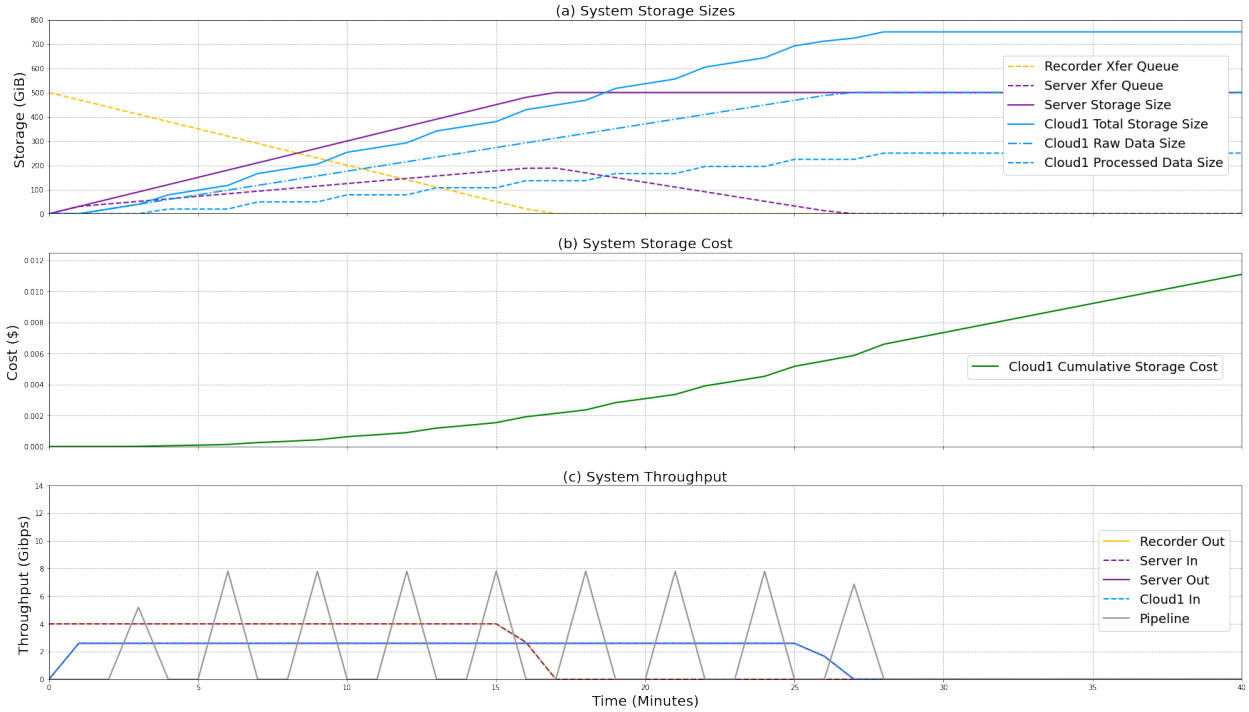

Figure 10. Scenario 2 Performance and Cost Results

The purple line in Figure 9(a) shows that scenario 1 server storage size and the server transfer queue remain near zero. This corresponds with the higher server throughput and the decision to delete server data after it is transferred to the cloud (rule 1). Figure 10(a) on the other hand, shows that scenario 2 server storage queue gradually increases over time as it uploads data to the cloud at a slower rate. This happens until all recorder data has been transferred at which point the server queue slowly diminishes. This figure also shows that scenario 2 server storage size increases to 500 GiB of data since it does not delete any data after it uploads it to the cloud (rule 1).

Figures 9(b) and 10(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 9(c) and 10(c) plot the throughput for the recorder, server, cloud, and 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 9(a) and 10(a) show that scenario 1 took longer to finish processing the data than scenario 2. However, Figures 9(b) and 10(b) show that scenario 1 cost less than scenario 2 by the end of the 40 minute simulation.

Discussion

The results in the previous section reveal at least one emergent system behavior that is not immediately apparent when comparing the two scenario configurations. Initially it appears that scenario 1 should be the fastest of the two, however the simulation results indicate otherwise. Specifically, Scenario 1 can have up to 5 pipelines running every 5 minutes, whereas scenario 2 can only have up to 4 pipelines every 6 minutes. This emergent system behavior shows that the data transfer rate from the server limits the data rate at which data 'builds-up' in the cloud processing queue. This smaller queue size between pipeline executions limits the number of parallel pipelines needed to meet the demand. This means that scenario 1 pipeline did not use all 5 parallel pipelines. Therefore, the longer time interval of scenario 1 reduced the system's overall speed. This is a result of the combination of pipeline interval timing (rule 2), maximum number of parallel pipelines (rule 3), and the rate of data transfer from the server to the cloud.

Figures 9(b) and 10(b), show that the slopes of the cost accumulation lines are noticeably different which indicates a higher cost rate for scenario 2. This intuitive result is from deleting data after it is processed (rule 4). As mentioned earlier, this is a cost vs. risk decision and we now have the estimated cost difference for cloud storage during the 40 minute simulation. This cost can now be weighed against the risks of deleting data and will enable informed decisions about the design. These results are nominally validated against cost estimations based on data from AWS, however this system model and simulation capability should be validated properly against time-series system data once it is collected.

One limitation of this approach we observe is upfront cost to build a system specification model and the simulation functionality in parallel. However, we believe upfront cost to build the foundational

SOA layers 1 and 2 will be offset by the speed of scaling this model to a larger size or more complex system later. We also assume the additional simulation capabilities the model provides and the reduction in time needed to integrate multiple toolsets will further offset this upfront cost.

Conclusions

The results of this demonstration show that our MBSE approach creates a time-dependent and simulation capable system specification model which accurately captures the state of the system as a function of time. In addition to successfully estimating the performance and cost, this system specification model accurately captures the emergent system behavior of processing speed, as discussed in the 3 section, that is not immediately obvious from the two scenario configurations. Capturing emergent system behaviors indicate the system model can produce valuable system design and analysis capabilities that are the basis for system optimization algorithms. System optimization and the resulting analysis is one of the main reason for building our approach. Optimization techniques can extract Pareto-optimal solutions and dominant system characteristics which provide trade space information to system engineers to more efficiently balance the system's performance against its costs. For complex cloud-based systems this could save companies tens of millions of dollars a year in cloud storage and compute costs. These saving will continue to grow as the adoption of cloud resources across all industries continue to climb each year.

The modularity and scalability of this approach is derived by the adoption of the layered SOA concept. Additionally, the modularity of this model and, by extension, this approach is shown by the capability to reuse these specification and simulation layers throughout the model and continue to produce valuable results. The ability to easily scale a small model up to one that is more complex or larger can reduce the time and effort needed when compared to an MBSE approach that does not scale well. The modularity of the specification model, also a result of our approach, can further reduce system engineering time and effort when troubleshooting issues with the model. Saving system engineering time and effort reduces system design and analysis costs.

The addition of model specification and simulation viewpoints allow system modelers and model users to understand the designed system. Clear and understandable system specifications can result in systems that are more likely to have fewer issues with system integration which also saves time and money.

Future Work

This MBSE approach is developed for system optimization approaches, therefore future work will focus on using Particle Swarm Optimization algorithms to produce Pareto-optimal solutions and extract Fuzzy Pareto Frontiers for this system and more complex cloud-based data systems. Additional work will also focus on refining our approach and applying it to expand the example system model to include, but not limited to: egress throughput and cost, ETL compute costs, Kubernetes (K8s) cluster baseline costs, Create, Read, Update, Delete (CRUD) costs, and stochastic network throughput.

Acknowledgments

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.

References

Broy, M. (2003). Service-oriented systems engineering: Modeling services and layered architectures. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *2767*, 48–61. doi. 10.1007/978-3-540-39979-7{_}4

Calheiros, R. N., Ranjan, R., Beloglazov, A., & Rose, A. F. D. (2011). CloudSim : a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. (August 2010), 23–50. doi. 10.1002/spe

Carroll, E. R., & Malins, R. J. (2016). *Systematic Literature Review: How is Model-Based Systems Engineering Justified?* (Tech. rep.). Sandia National Laboratories. http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

Cho, K., & Bahn, H. (2020). A Cost Estimation Model for Cloud Services and Applying to PC Lab Platforms.

Douglass, B. P. (2016). *Agile systems engineering*.

FinOps Foundation. (2023). FinOps Organization. https://www.finops.org/

Flexera. (2023). *2023 State of the Cloud Report* (tech. rep.). Flexera. https://info.flexera.com/CM-REPORT-State-of-the-Cloud?lead_source=Website%20Visitor&id=Blog

Graves, H., Guest, S., Vermette, J., Bijan, Y., Banks, H., Whitehead, G., & Ison, B. (2009). *Air Vehicle Model-Based Design and Simulation Pilot* (tech. rep.).

Herzig, S. J., Mandutianu, S., Kim, H., Hernandez, S., & Imken, T. (2017). Model-transformation-based computational design synthesis for mission architecture optimization. *IEEE Aerospace Conference Proceedings*. doi. 10.1109/aero.2017.7943953

Karban, R., Jankevičius, N., & Elaasar, M. (2016). ESEM: Automated Systems Analysis using Executable SysML Modeling Patterns. *INCOSE International Symposium*, *26*(1), 1–24. doi. 10.1002/j.2334-5837.2016.00142.x

LaSorda, M., Borky, J. M., & Sega, R. M. (2018). Model-based architecture and programmatic optimization for satellite system-of-systems architectures. *Systems Engineering*, *21*(4), 372–387. doi. 10.1002/sys.21444

Ryan, J., Sarkani, S., & Mazzuchi, T. (2014). Leveraging Variability Modeling Techniques for Architecture Trade Studies and Analysis. *Systems Engineering*, *17*(1), 10–25. doi. 10.1002/sys

Wickremasinghe, B., Calheiros, R. N., & Buyya, R. (2010). CloudAnalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications. *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*, 446–452. doi. 10.1109/aina.2010.32

Biography

Mr. Thomas M. Booth is the System Architect and Integration Director for a research and development team within the 309th Software Engineering Group at Hill AFB. Among many other projects, this team is focused on data analytics, data pipelines, DevSecOps SW development, and data mesh product development. Furthermore, this team is refining MBSE approaches and techniques for complex, software-based systems towards data driven system design, analysis, and optimization. He received his BS in Mechanical Engineering (ME) from the University of Utah (UofU) in 2003, his MS in ME from the UofU in 2012, passed his Ph.D. preliminary exam in Systems Engineering from Colorado State University in 2023, and is currently working on his Ph.D. dissertation. His current research interests focus on leveraging SysML for Pareto-optimal and Fuzzy Pareto Frontiers towards design analyses for complex software-based cloud data systems. His previous experience includes: USAF avionics software system integration and test director, NASA CFD and wind tunnel test engineer, and USAF aero-performance engineer.

Dr. Sudipto Ghosh is a Professor of Computer Science at Colorado State University with an affiliate appointment in Systems Engineering. He received the Ph.D. degree in Computer Science from Purdue University in 2000. His research interests are in software engineering (design and testing). He is on the editorial boards of IEEE Transactions on Reliability, Software and Systems Modeling, Software Quality Journal, and Information and Software Technology. Previously he was on the editorial board of the Journal of Software Testing and Reliability. He was a general co-chair of MODELS 2009 (Denver) and Modularity 2015 (Fort Collins). He was a program co-chair of ICST 2010 (Paris), DSA 2017 (Beijing), ISSRE 2018 (Memphis), ISEC 2024 (Bangalore), and QRS 2024 (Cambridge). He has served on program committees of multiple conferences. He is a member of the ACM and a Senior Member of the IEEE.