Traceable System of Systems Explorations Using RCE Workflows

Jorge Lovaco[a,*], Ingo Staack[b], Petter Krus[a]

[a] *Department of Management and Engineering (IEI), Linköpings university, Sweden,* [b] *Institute of Aircraft Design and Lightweight Structure (IFL), Technische Universität Braunschweig.*
jorge.lovaco@liu.se

Abstract

The System of Systems (SoS) framework plays a pivotal role in delimiting aircraft design spaces by examining interactions among its Constituent Systems (CS). Each CS has a distinct collection of capabilities, some of which may be shared with other CS. The framework explores emergent behaviours that arise from communication between the CS within the SoS. These emergent behaviours are characterized by their unattainability by any individual CS and result from their collaborative nature. The identification of these emergent behaviours enables System of Systems Engineering (SoSE) to pinpoint the most valuable configurations of the SoS, thereby maximizing the collective value. Furthermore, these emergent behaviours aid in stipulating design requirements for new systems based on the capabilities outlined in the SoS study. To map the relationship between needs, capabilities, requirements, and behaviours, maintaining traceability throughout the study is paramount.
This research employs workflows created using the Remote Component Environment (RCE), a specialized tool for structured and automated task development. The objective is to showcase RCE's integration capabilities- specifically for software tools and Python scripts- with task scheduling. This integration enables swift extraction of results, making them available at every step, thus augmenting analysis efficiency. The study focuses on the perspective of an aircraft designer during the early concept generation phase, specifically applied to the development of an electric Unmanned Aerial Vehicle (UAV) concept for wildfire detection.
Keywords: System of Systems, Aircraft Conceptual Design, Wildfire Detection, Agent-Based Simulation, Systems Engineering.

1. Introduction

The System of Systems (SoS) analysis has become an important part of systems engineering (Staack, 2019). An SoS is defined as a system that consists of several Constituent Systems (CS) and shares the following characteristic properties (Maier, 2014):

- Each of the CS can operate by itself and execute its mission as an individual.
- Each of the CS is managed independently, which means that each one of them is acquired and maintained within its own budget constraints.
- The CS are under a geographic distribution, with the distance large enough to require a non-physical communication.
- Communication and interaction between CS results in emergent behaviours, which can be not only difficult to predict but also unattainable by any single CS.
- There is an evolution throughout time of the SoS, either by adding or removing CS, upgrading or renewing them, or integrating CS different from the ones in the original composition.

Modelling and simulation become an obvious choice for finding emergent behaviours in an SoS configuration and to study its evolution when the CS arrangement changes.
With an SoS being a complex assembly of systems, large numbers of possible combinations might be found during the different analyses. Thus, the requirement spaces must be meticulously managed to prevent confusion and ensure traceability at every step of the process (Mori, 2018). Traceability ensures that the customer needs are correctly understood and linked to requirements and use cases (Luzeaux & Ruault et al., 2013). Requirement traceability also supports greatly the analysis of an SoS. It narrows down the design space and enhances trade-off analysis (Staack, 2019), from the top level of the SoS, down to the subsystem level of the CS in an asynchronous manner (Dahmann, 2008). Poor traceability makes it difficult to identify purposes or goals resulting in the less convenient "*bureaucracy-driven architectures*" (Maier, 2014). Enhancing traceability helps also to propagate changes among

different levels, namely concepts, requirements, specifications as well as take decisions or perform better impact studies. Traceability involves matrix-based methods (Krus, 2006) for mapping customer needs and requirements, design parameters and system behaviour. These methods allow re-evaluations in case of requirement changes (Luzeaux & Ruault et al., 2013). Besides traceability-oriented methods, it is possible to find requirements for traceability of SoS capabilities (Tekinerdogan, 2017).

A wildfire detecting SoS is used in the present work to illustrate some of the modelling and simulations considerations needed to develop traceable workflows. The scenario map, its environmental conditions, and the CS included must be allowed to change or evolve to study the SoS under different conditions. The outcome of the simulated scenarios is to be analysed for obtaining a set of requirements to constrain the design space of an electric Unmanned Aerial Vehicle (UAV). For the SoS explorations, the feasibility of Remote Component Environment (RCE) workflows will be evaluated with a focus on traceability. This paper aims to assess the convenience of using RCE's workflows in terms of traceability.

2. A Workflow-based Modelling Environment

The Remote Component Environment (RCE) is an open-source application developed by the German Aerospace Center (DLR) for design and simulation of systems (Boden, 2021). Serving as a flexible and scalable platform, it utilizes object-components, which represent a series of tasks to be executed in a specific order at predetermined intervals. RCE provides a graphical user interface for configuring the workflows, empowering users to define inputs, outputs, and the sequence of individual tasks. This structure ensures systematic and efficient experiments.

2.1. RCE Components

RCE features a range of components for different functions such as simulation, data processing and visualization. Central to these components available in RCE is the Design of Experiments (DoE) component, which allows users to define sets of input parameters with specific ranges and distributions, to set up and run experiments.

RCE also supports the incorporation of Python scripts as workflow components. This feature leverages Python's versatility, extending the range of tasks beyond the capabilities of the default RCE components. For example, conduct intricate calculations, manipulate data or files, or generate plots during the workflow runtime.

Additionally, RCE enables the integration of external tools as components in its workflows. This means that users can add their own software packages to RCE- for instance, by adding external tools for data acquisition, simulation, or analysis- and incorporate them into a workflow with built-in components for optimization.

2.2. System of Systems Experiments

Conducting SoS experiments for Aircraft Conceptual Design (ACD) is a complex application that requires the integration of multiple components for considering the CSs and their respective subsystems. The SoS experiments aim at optimizing requirements for a new aircraft, considering factors such as performance, safety, and cost. To achieve this, the DoE component within RCE is used to orchestrate and execute a series of experiments that delve into the compromises inherent between varying requirements and capabilities. The results of these experiments can then be analysed using Python scripts and external tools to identify the most promising design options. An example of an SoS workflow is shown in Fig. 1.

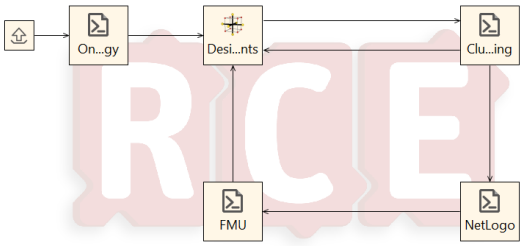

Figure 1: Structured workflow in RCE for SoS experiments.

3. SoS Workflow Construction

This section describes the different disciplines involved as block components in the assembly of the RCE workflow for wildfire detecting SoS exploration. Detection hinges on the subsystems nested within the CS. For this study, these subsystems encompass visual sensors with a resolution range. When a smoke plume is captured within the sensor's visual cone, detection is confirmed, marking the SoS's operation as successful and stopping the simulation.

3.1. Ontology Modelling

Ontologies serve as an instrument to formally encapsulate knowledge specific to a domain, including the concepts, relationships, and constraints that define it (Knöös Franzén, 2023). They provide a shared vocabulary and understanding for a group of people working on a common task, enabling more effective communication and reasoning about the domain. By defining a common ontology of the components, interfaces, and behaviours of the systems in the

study, it becomes easier for everyone involved to reason about how different systems interact with one another, and how changes done to one system will affect the others. Additionally, ontologies can facilitate the tracing of requirements, decisions, and outcomes across the SoS, which is important for understanding the impact of changes and making informed decisions (Lovaco, 2023). For visual clarity, Fig. 2 shows the ontology structure of a surveillance UAV. One ontology definition example (used for defining the present scenario) can be seen in Fig. 3.

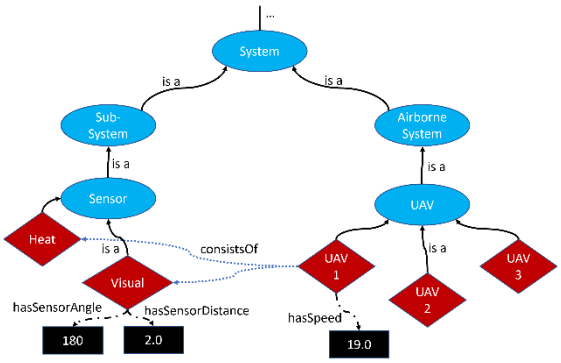

Figure 2: Partial ontology structure of a surveillance UAV.

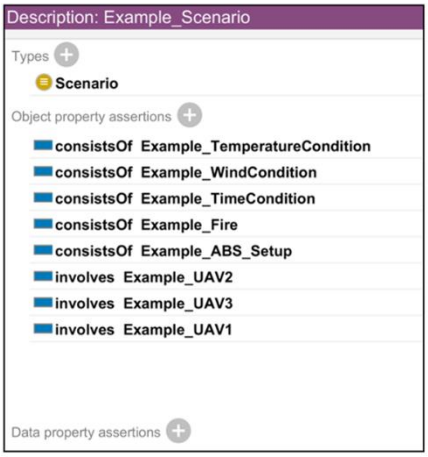

Figure 3: Ontology definitions of a surveillance UAV.

3.2. Clustering

Clustering can be described as the process of grouping a set of objects in such a way that objects in the same group (called a cluster) are in some manner more alike than those in different groups (clusters). Clustering is needed for navigation purposes since the flight paths will be defined afterwards based on the clusters to be visited.

The K-Means clustering algorithm is a popular method for clustering (Pedregosa, 2011). It partitions *N* data points into *K* clusters gauging their proximity to the centroids. Fig. 4 shows an example of the centres generated after clustering a given data set. However, determining the optimal number of clusters often poses a challenge. The method for determining the optimal number of clusters in K-Means clustering is explained in the section below.

3.2.1. Objective Function

The objective function of K-Means is to minimize the sum of squared distances between each data point and its assigned centroid:

$$\sum_{i=0}^{n} min_{\mu_j \in C} \left\| x_i - \mu_j \right\|^2 \qquad (1)$$

The time complexity of K-Means is O(KNT), where *N* is the number of data points and *T* the iteration number (Pedregosa, 2011).

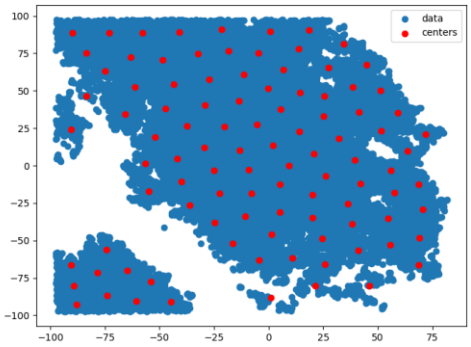

Figure 4: Centre points after clustering data.

A method for estimating the optimal number of clusters involves plotting the Within-Cluster Sum of Squares (WCSS) against the number of clusters and selecting the point where the WSS starts to level off (Pedregosa, 2011). The WCSS is defined as the sum of squared distances between each data point and its assigned centroid:

$$WCSS = \sum_{C_k}^{C_n} \left(\sum_{d_i in C_i}^{d_m} distance(d_i, C_k)^2 \right) \qquad (2)$$

Where *C* is the cluster centroid, *d* the data point and *n* the number of clusters.

The optimal number of clusters can be determined by inspecting the plot of WCSS as a function of the number of clusters, shown below in Fig. 5. Typically, the plot will show a steep decrease in WCSS as the number of clusters increases, followed by a levelling off. The "Elbow point" is the number of clusters at which the WCSS starts to level off. For the plot shown below, approximately 5 clusters yield a value of $0.2 \cdot 10^8$.

While visually perusing these plots is feasible, it might prove tedious and suboptimal, especially in large-scale simulations. In contrast, the use of a convergence analysis provides an automated and systematic approach. This allows for quicker identification of convergence behaviour and facilitates the exploration of design spaces in a more efficient and reliable manner.

A straightforward, yet more automation-friendly approach is to take the computed sum of squared distances and average it using the amount of data collected in that group. The result is compared with the squared value of a convenient parameter, which for the present case is the squared value of the distance that the sensors in the UAV can see.

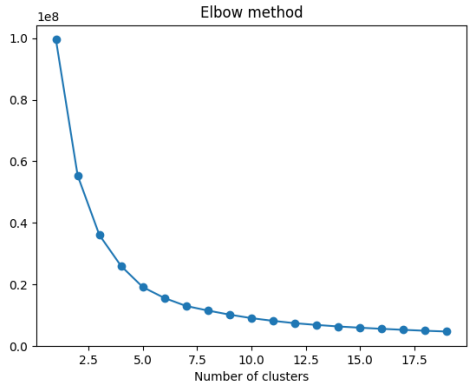

Figure 5: Evolution of the inertia depending on the number of clusters used.

3.3. Graph Theory for Finding the Shortest Navigation Path

The Traveling Salesman Problem (TSP) is a well-known combinatorial optimization problem that involves finding the shortest path to visit a set of points exactly once and return to the origin (Reinelt, 2003). One prevailing technique to tackle the TSP is by using graph theory (Euler, 1741), which provides an efficient and scalable solution (Hagberg, 2008). In this method, the TSP is modelled as a complete graph, where each point is represented by a node, and the edges represent the distances between them. The objective is to find the Hamiltonian cycle, which is a path that visits all nodes exactly once and returns to the starting node. Various algorithms, such as Christofides algorithm (Christofides, 1976), Simulated Annealing (Kirkpatrick, 1983), and Threshold Accepting (Dueck, 1990), can be used to find the shortest path. By using graph theory, the TSP can be solved with high accuracy, making it a useful tool for solving optimization problems in various fields. For the SoS use case, the centroids found in the previous step are clustered again depending on the number of CSs to be used, which for the present case is the number of UAVs. The TSP is solved for each aircraft to generate the navigation path for patrolling over an area. The Christofides algorithm is the chosen one for the present work, and it is used over a complete graph *G* to generate paths such as the one shown in Fig. 6. The initial node for the path is 0, but the nodes are not necessarily visited in the same order as they are numbered. The Christofides algorithm consists of the following steps (Goodrich & Tamassia et al., 2015):

1. A minimum spanning tree, *M*, is constructed for *G*.
2. Compute the set *W* with the vertices with odd degree from *M*. Form a new graph *H* with these vertices and the edges connecting them in *G*. Compute a minimum-weight matching *P* in this subgraph *H*.
3. Then *H* and *P* are combined into *G'* keeping repeated edges.
4. Find an Eulerian path *C*.
5. Finally convert *C* into a tour by skipping each vertex that has already been visited.

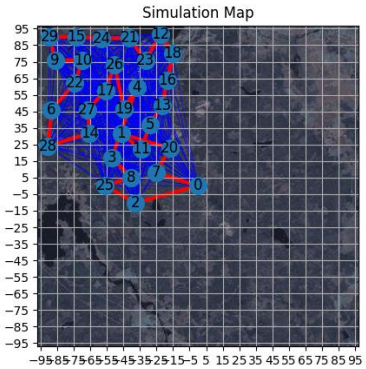

Figure 6: Hamiltonian cycle generated from a TSP solution.

3.4. Agent-Based Simulations for SoS Exploration

Agent-Based simulations (ABS) are a distinctive modelling and simulation technique that focuses on the behaviour of individual agents. Agents are entities with the ability to perceive their environment, make decisions, and act on their environment.

ABS boasts several advantages when compared to other modelling and simulation techniques. For one, ABS allows for the modelling of heterogeneity among agents, they can have different attributes, behaviours, and interactions, which makes the system more realistic (Lovaco, 2022). Moreover, ABS also allows to observe emergence, which is the phenomenon where the behaviour of the SoS is unequal to the sum of its individual parts. It must be noted that the resulting emergent behaviour should be judged as an increase or decrease of the SoS group value. Consequently, ABS is a promising approach to exploring the behaviour of interacting systems.

The exploration of complex SoS requires sophisticated tools for modelling and simulation. The tool used for this paper is NetLogo, an open-source ABS software with a drag-and-drop interface to create Agent-Based Models (ABM) (Tisue, 1999). Fig. 7 shows a NetLogo model interface created for firefighting SoS studies. NetLogo has several built-in capabilities for creating and manipulating agents, defining their behaviours, and visualizing simulation results.

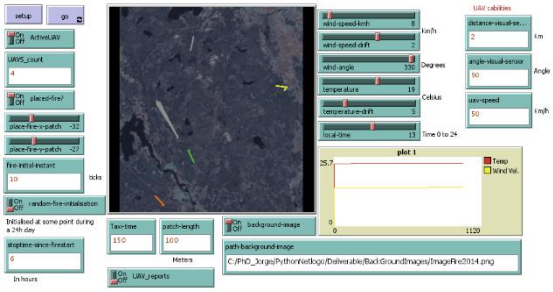

Figure 7: Agent-Based Model Interface.

3.5. Functional Mockup Units

The results extracted from the ABS can be analysed to find, for example, range and endurance requirements for UAVs. Once these specifications are distinctly outlined, cyber-physical UAV models with higher fidelity than the ones defined in the ABS are more suitable for ACD. These high-fidelity models can be made tool independent by using Functional Mockup Units (FMUs). Gaining traction in contemporary engineering and modelling sectors, FMUs are software modules that enshroud a specific functionality of a larger model of a system (Blochwitz, 2011). The main objective of FMUs is to enable model share and exchange between different tools and platforms in a standardized way. This means that models developed in one software environment can be easily integrated and used in another, without the need for custom integration code. FMUs are designed to be platform and language independent, they can be used across different operating systems and hardware architectures. This positions FMUs as a pivotal asset to share work and collaborate on interdisciplinary complex modelling and simulation projects (Braun, 2013).

4. SoS Case study and Results

This section presents the case study of a homogeneous forest-fire spotting SoS that consists exclusively of UAVs patrolling a given area to detect wildfires. Their respective flight trajectories, however, are derived uniquely by harnessing the K-Means clustering algorithm—which groups proximate trees—and subsequently deploying the TSP to delineate the Hamiltonian cycle for every UAV. Clustering is performed using the radius around each centroid determined by the visual sensor proficiency of the UAV. To evaluate different SoS configurations, the DoE uses a Latin Hypercube Sampling (LHS) method to generate different CS for the scenario. The UAV capabilities and scenario parameters not generated using LHS are extracted from the ontology, which is stored in an XML file. The simulations are executed in Netlogo with the wildfire initial geographic position being at a random point in the map. The fire detection success or failure is reported for each scenario. The flight missions for each UAV are simulated using a high-fidelity model FMU to evaluate the State of Charge (SoC) of the batteries and determine if the UAV configuration can achieve their mission (Krus, 2012). The workflow diagram of this case study was shown in Fig. 1, which from left to right starts by loading and reading the ontology; then the DoE component generates the different CS; the vegetation data is clustered afterwards to generate the UAVs flight paths; the NetLogo ABS executes the different SoS configurations; finally, the high-fidelity FMUs are executed to evaluate the performance of each aircraft concept. The workflow cycle is repeated as many times as initially defined in the DoE component. All the outputs forwarded at each step of the workflow are stored by RCE and accessible for the user once the experiment is completed, which is key for traceability.

4.1. Experiment Workflow

An ontology akin the represented in Fig. 2 is used to describe the scenario. The ontology XML file is read to extract the information needed for the SoS experiments. For this paper the values extracted from the ontology are the scaling factor, the fire detection time limit, position of the fire, wind velocity and wind direction. The number of UAVs used, and their capabilities are generated using the LHS and are catalogued in Tab. 1. This data is pivotal in tailoring the ABS, enabling a thorough examination of the ramifications stemming from diverse SoS layouts.

Table 1: LHS Generated Experiments.

ID	UAVS	Sensor Range [km]	Velocity [km/h]
1	6	1	88
2	7	2	106
3	9	1	153
4	4	2	190

Navigation routes are formulated in alignment with methodologies delineated in Sections 3.2 and 3.3. The paths depend on the sensor visual range and the quantity of UAVs defined. Fig. 4 shows the discerned centroids and Fig. 6 offers a glimpse into one such navigation route. The nondimensional ranges computed are shown in Tab. 2. The “Min Range” and “Max Range” values in the table represent respectively the minimum and maximum nondimensional distance flown by the UAVs in the SoS configuration. The scaling factor aids in converting these nondimensional distances into SI units.

Table 2: Nondimensional Ranges and Map Scaling Factor.

ID	Min Range	Max Range	Scaling Factor [m]
1	329	465	100
2	278	454	100
3	224	346	100
4	364	623	100

The ABS is performed for a map, as depicted in Fig. 8, where different surveillant agents can be seen. The map geography is generated importing elevation points into the model. The vegetation is dispersed randomly under certain topographic criteria, such as lake zones, resulting into the data distribution shown previously in Fig. 4. The geography and the fire location intend to relate to a real fire case (MSB, 2014). The arrow in the figure indicates the source of the smoke plume and the coloured lines represent a fraction of the flight paths taken by the aircraft systems.

Figure 8: Agent-Based simulation visualization.

The scenario was simulated using a quartet of different SoS configurations, each of which was executed ten times with wildfires initialized randomly, accumulating a total of 40 runs. Tab. 3 outlines the detection success rate and the averaged time required to spot the smoke plume for each SoS configuration.

Following the workflow sequence from Fig. 1, a high-fidelity aircraft model available in the cyber-physical high-performance simulation tool Hopsan (Krus, 2012) (which is similar to e.g., Modelica) was simulated through an FMU. This was done to ascertain the SoC of the battery, as well as the propeller diameter required to achieve the necessary velocity.

Table 3: Detection success and time.

ID	Success Rate [%]	Avg. Detection Time[min]
1	100	92
2	100	139
3	90	128
4	90	70

The necessary endurance, corresponding to the most extensive range for each UAV setup, is calculated and used to define the stop time for each FMU simulation. Tab. 4 summarizes the results extracted from these simulations: the battery SoC, the required diameter for the propeller, and the required endurance to complete just one loop of the predefined flight path.

Table 4: Aircraft FMU values.

ID	SoC [%]	Propeller Ø [cm]	Endurance [min]
1	73.0	28	32
2	78.2	28.5	26
3	86.5	31.5	14
4	78.1	34	20

5. Summary and Discussion

The workflow created ran all the 40 cases to study a firefighting SoS in 39 minutes real time. The obtained results can be used for constraining the design space to generate aircraft concepts, as well as discussing the trade-offs needed to achieve the capabilities of the desired aircraft concept. For a good trade-off analysis, a more extensive set of results is needed. But the procedure is exemplified in this section. First, Tab. 5 shows the collection of different CS capabilities.

Table 5: Constituent Systems Specifications.

CS	Sensor Range [km]	Velocity [km/h]	Range [km]	Prop. Ø [cm]	Endurance [min]
1	1	88	46.5	28	32
2	2	106	45.4	28.5	26
3	1	153	34.6	31.5	14
4	2	190	62.3	34	20

Then a comparison of SoS capabilities with their success rates (Tab. 4) helps in ranking requirements. This ranking can be subjective, based on the success rates, or on customer pre-defined criteria, culminating in a priority matrix. In Tab. 6, values 2, 1, or 0 denote higher, equal, or lower importance. Notably, below the diagonal, values swap between 0 and 2, while 1 stays constant. The trade-offs, discussed by rows are as following: Sensor Range (SR) holds higher significance than CS Velocity (V) since a higher speed does not increase the success;

comparison of SR with Range (R) or Endurance (E) needs more data for a decision hence it is assumed equal importance; SR, however, is seen as more crucial than Propeller Diameter (Ø) for fire detection.
The comparison is only exemplified here since it is beyond the current aim. It needs to be mentioned that, after identifying and discussing trade-offs to prioritize requirements, the ranking is eventually combined with methods such as the House of Quality (HoQ) or Quality Function Deployment (QFD) (Ulrich & Eppinger et al., 2016).

Table 6: Customer Requirement Priorities.

	SR [km]	V [km/h]	R [km]	Ø [cm]	E [min]
SR		2	1	2	1
V	0		0	2	1
R	1	2		2	2
Ø	0	0	0		1
E	1	1	0	1	

Fig. 9 shows an approximate curve representing the achieved cruise velocity as a function of the propeller diameter for a specified aircraft body geometry, which once they are obtained it is possible to find feasible regions for the design spaces.

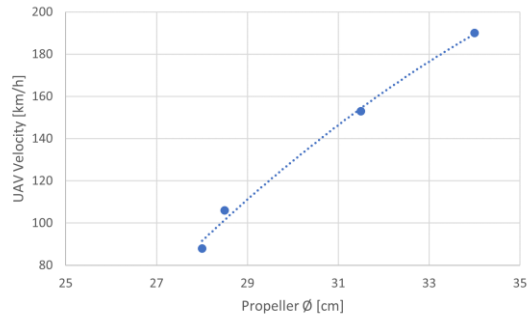

Figure 9: Achieved UAV cruise velocity as a function of the propeller diameter.

From the analysis, a notable topic is the aircraft's taxiing time. As indicated in Tab. 4, the aircraft can complete several missions along its flight path before needing to taxi, with the goal of preserving a SoC above 20% ideally. If this is not achieved, there might be a need to redesign the battery and aircraft to lessen their weight. This reduction can decrease both cruise speed and associated costs, potentially prompting another iteration of ACD.

For battery sizing, the drag forces over the mission can be extracted from the FMU results as well. Fig. 10 shows the flight altitude, and the aircraft drag over mission time. By knowing the altitude, it is possible to obtain the atmosphere properties. By understanding the forces that the aircraft needs to overcome to sustain the flight, it is possible to obtain the work and energy needed to fulfil the mission, and thus size a battery accordingly. Furthermore, studying the drag forces can reveal the parts of the geometry with the higher contribution to them and initiate a subsystem optimization process. These results can be obtained thanks to the use of a high-fidelity FMU in the workflow. It helps to explain, for example, the reduction over time of the drag forces observable in Fig. 10, which can be attributed to the decreasing energy in the battery that is translated into a reduced cruise velocity over the mission time.

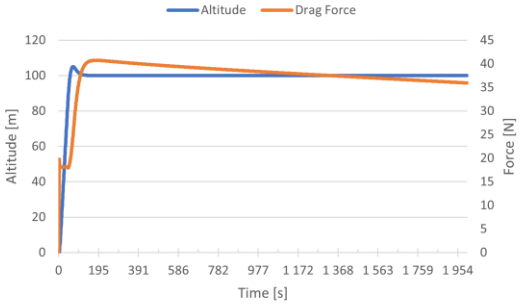

Figure 10: Aircraft altitude and drag forces values throughout time.

6. Future Work

The SoS studies for ACD can be expanded through the compatibility of RCE with XML files and CPACS files (Alder, 2020). CPACS files are designed for storing information related to aircraft concept generation. They can be used for encapsulating data pertinent to aircraft concept development, present a means to consolidate and disseminate expertise across diverse engineering domains, encompassing aerodynamics, propulsion, and structures.
RCE is equipped with dedicated features for the integration and utilization of other CPACS compatible tools in its workflows. One example of a tool designed to work seamlessly with CPACS files is TiGL (Siggel, 2019). TiGL is open-source and uses geometry libraries to visualize the 3D geometric representations of data extracted from CPACS. TiGL is also capable to export to several computer-aided design (CAD) formats.
Another promising future inquiry hinges on multidisciplinary optimization. The results of the SoS exploration studies can be analysed using HoQ and QFD methods for creating SoS value and system cost functions from the requirements. Subsequently analysing the balance between SoS value and CS cost to set the stage for a clearly delineated design space, marked by discernible requirement constraints, primed for preliminary concept evolution and optimization.

7. Conclusion

The research presented here using RCE workflows offers a robust method for tracing requirements and capabilities in SoS studies. A case study for aircraft concept generation to be used for wildfire detection was introduced to illustrate the created workflow. From parameters generated using LHS, navigation routes were generated from areas clustered based on the UAVs visual subsystems capabilities. The aircraft concepts and their routes were simulated to compare SoS mission success rates and constrain the design space of the aircraft concepts. Overall, this work illustrates how RCE workflows can be used for aircraft requirement generation by providing easy access to the results of every experiment allowing to sustain a high level of traceability.

Acknowledgment

The authors would like to acknowledge the Swedish Innovation agency (VINNOVA) for its financial support through the grant 2019-05371.

References

Mori, M., Ceccarelli, A., Lollini, P., Frömel, B., Brancati, F., & Bondavalli, A. (2018). Systems-of-systems modeling using a comprehensive viewpoint-based SysML profile. Journal of Software: Evolution and Process, 30(3), e1878.

Staack, I., Amadori, K., & Jouannet, C. (2019). A holistic engineering approach to aeronautical product development. The Aeronautical Journal, 123(1268), 1545-1560.

Dahmann, J. S., & Baldwin, K. J. (2008, April). Understanding the current state of US defense systems of systems and the implications for systems engineering. In 2008 2nd Annual IEEE Systems Conference (pp. 1-7). IEEE.

Maier, M. W. (2014). The role of modeling and simulation in system of systems development. Modeling and simulation support for system of systems engineering applications, 11-41.

Krus, P. (2006, September). Aircraft System Optimization and Analysis for Traceability in Design. In 11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference (p. 7017).

Tekinerdogan, B., & Erata, F. (2017, April). Modeling traceability in system of systems. In Proceedings of the Symposium on Applied Computing (pp. 1799-1802).

Blochwitz T., Otter M., Arnold M., Bausch C., Clauß C., Elmqvist H., Junghanns A., Mauss J., Monteiro M., Neidhold T., Neumerkel D., Olsson H., Peetz J.-V., Wolf S. (2011): The Functional Mockup Interface for Tool independent Exchange of Simulation Models. 8th International Modelica Conference, Dresden 2011.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. the Journal of machine Learning research, 12, 2825-2830.

Hagberg, A., Swart, P., & S Chult, D. (2008). Exploring network structure, dynamics, and function using NetworkX (No. LA-UR-08-05495; LA-UR-08-5495). Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

Lovaco, J., Staack, I., & Krus, P. (2022). Environmental Agent-Based Modelling For A Firefightingsystem Of Systems. In ICAS2022.

Lovaco, J. L., Franzén, L. K., & Krus, P. (2023, April). Agent-Based Simulation and Ontology Integration for System-of-System Exploration. In Proceedings of IDEAS 2022: Interdisciplinary Conference on Innovation, Design, Entrepreneurship, and Sustainable Systems (pp. 13-23). Cham: Springer International Publishing.

Axin, M., Braun, R., Dell'Amico, A., Eriksson, B., Nordin, P., Pettersson, K., ... & Krus, P. (2010). Next generation simulation software using transmission line elements. In Fluid Power and Motion Control, 15th-17th September, Bath, England, UK (pp. 265-276). Centre for Power Transmission and Motion Control.

Modelica Association. Modelica - A unified ObjectOriented Language for Physical Systems Modeling Language Specification - Version 3.2, 03 2010.

Krus, P., Braun, R., Nordin, P., & Eriksson, B. (2012). Aircraft system simulation for preliminary design. In 28th International Congress of the Aeronautical Sciences, Brisbane, Australia, 23-28 September, 2012 (pp. Art-nr). Optimage Ltd.

Braun, R., & Krus, P. (2013). Tool-independent distributed simulations using transmission line elements and the Functional Mock-up Interface. In 53rd SIMS conference on Simulation and Modelling, October 4-6, Reykjavik, Iceland.

Alder, M., Moerland, E., Jepsen, J., & Nagel, B. (2020). Recent advances in establishing a common language for aircraft design with CPACS.

Siggel, M., Kleinert, J., Stollenwerk, T., & Maierl, R. (2019). TiGL: an open source computational geometry library for parametric aircraft design. Mathematics in Computer Science, 13(3), 367-389.

Boden, B., Flink, J., Först, N., Mischke, R., Schaffert, K., Weinert, A., ... & Schreiber, A. (2021). RCE: an integration environment for engineering and science. SoftwareX, 15, 100759.

Myndigheten för samhällsskydd och beredskap (MSB). Skogsbranden i Västmanland 2014. MSB798, 2015. ISBN: 978-91-7383-527-5

Tisue, S., & Wilensky, U. (1999). Center for Connected Learning and Computer-Based Modeling Northwestern University, Evanston, Illinois. NetLogo: A Simple Environment for Modeling Complexity, Citeseer.

Christofides, N. (1976). Worst-case analysis of a new heuristic for the travelling salesman problem. Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group.

Dueck, G., & Scheuer, T. (1990). Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. Journal of computational physics, 90(1), 161-175.

Kirkpatrick, S., Gelatt Jr, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. science, 220(4598), 671-680.

Knöös Franzén, L. (2023). A System of Systems View in Early Product Development: An Ontology-Based Approach (Doctoral dissertation, Linköping University Electronic Press).

Euler, L. (1741). Solutio problematis ad geometriam situs pertinentis. Commentarii academiae scientiarum Petropolitanae, 128-140.

T. Reinelt, G. (2003). The traveling salesman: computational solutions for TSP applications (Vol. 840). Springer.

Goodrich, M. T., & Tamassia, R. (2015). Algorithm design and applications (Vol. 363). Hoboken: Wiley.

Luzeaux, D., & Ruault, J. R. (Eds.). (2013). Systems of systems. John Wiley & Sons.

Ulrich, K. T., & Eppinger, S. D. (2016). Product design and development. Boston: McGraw-Hill higher education.