CEAS Space Journal
https://doi.org/10.1007/s12567-025-00626-7

ORIGINAL PAPER

System architecture optimization: an example application to space mission planning

Jasper H. Bussemaker[1] · **Thomas Firchau**[2]

Received: 27 August 2024 / Revised: 30 April 2025 / Accepted: 7 May 2025

Abstract
Space mission planning involves coupled architecture decisions that non-trivially influence system-level performance metrics such as weight, power usage, and scientific value. We present an exemplary space mission planning problem involving several mission-level and spacecraft-level choices, and optimizing for the conflicting objectives of system mass and scientific value. The problem is solved using System Architecture Optimization (SAO): a technique where numerical optimization algorithms are used to explore an architecture design space and find a Pareto front of optimal architectures. The architecture design space is modeled using the Architecture Design Space Graph (ADSG) implemented in the ADORE editing and optimization tool. The design space model includes function-component allocation choices, component-level design variables, and system-level objectives and constraints to optimize for. Evaluation code is implemented in Python and linked to the design space model using class factories. The design space is explored using NSGA-II, a multi-objective evolutionary algorithm, resulting in a Pareto front trading-off system mass and total experiment duration.

Keywords System architecture · MBSE · Mission planning · Optimization

1 Introduction

Increasing system complexity and more stringent stakeholder needs in the space domain require a shift towards innovative model-based design space exploration approaches in early design phases [1]. Such a shift would allow for exploring more potential concepts, at a higher level of detail and subject to less designer bias, earlier in the design phase [2]. In particular, design decisions early in the design process include selecting payloads such as scientific instruments, calculating operational budgets such as operational time and data handling, and sizing support systems such as power supply and thermal regulation systems. Design decisions are coupled and each influence system-level performance, such as cost, weight or scientific value, in conflicting and non-linear ways [3]. This makes the design of space missions a non-trivial task, one which might be supported by increased automation in the design space exploration to successfully balance trade-offs between conflicting goals.

In this paper, we demonstrate the application of System Architecture Optimization (SAO) techniques for space mission design. SAO enables exploring a larger number of potential candidates earlier in the design process by formulating the architecting process as a numerical optimization problem [4]. This results in improved decision-making due to a more complete overview of the design space and less bias towards conventional designs. Compared to traditional systems engineering approaches, SAO shifts focus from designing one particular solution to a problem, to the definition and development of the architecture design space model and performance evaluation code [5]. An architecture design space model defines the architectural decisions and how they influence each other, and enables automatic generation of architecture candidates by the optimization algorithm. The performance evaluation code enables automatic calculation

Jasper H. Bussemaker and Thomas Firchau have contributed equally to this work.

✉ Jasper H. Bussemaker
jasper.bussemaker@dlr.de

Thomas Firchau
thomas.firchau@dlr.de

[1] Institute of System Architectures in Aeronautics, German Aerospace Center (DLR), Hamburg, Germany

[2] Institute of Space Systems, German Aerospace Center (DLR), Bremen, Germany

Published online: 18 June 2025

of system-level performance measures for any architecture candidate.

We continue with an overview of SAO in Sect. 2. The space mission planning application case is described in Sect. 3. Results are presented and discussed in Sects. 4 and 5 concludes the paper.

2 System Architecture Optimization

System Architecture Optimization (SAO) involves the application of numerical optimization algorithms for designing system architectures. The architecture of a system is a central artifact in systems engineering, and specifies what components a system consists of (the elements of *form*), and how they collaborate to fulfill the system *functions* (i.e. what the system performs) [6]. Many potential architectures may exist for a given design problem, making it infeasible to exhaustively consider all architectures [7]. This warrants the application of optimization techniques to selectively search the design space. The latter is achieved by assigning components (i.e. elements of form) to functions. Functions are optimally specified in a solution-neutral manner, so that they can be used to identify technology (i.e. component) alternatives for fulfillment. Next to function fulfillment, designing a system architecture may also involve specialization and characterization of form, for example, by selecting the number of instances of components and finding the optimal values for component-specific design parameters, and connecting components, such as assigning power consumers to power sources or routing data connections.

Two elements are required to implement a system architecting process as an optimization problem: a way to generate architecture alternatives from an architecture design space model, the *architecture generator*, and a way to quantify the performance of each generated architecture to enable fair comparison, the *architecture evaluator* [4]. The architecture generator is driven by an optimization algorithm that generates a new design vectors, which get converted into architecture instances to be evaluated by the architecture evaluator. SAO problems in general have several salient features that make solving them challenging [8]. Design variables might both be discrete (e.g. architectural choices) and continuous (e.g. component sizing parameters). Design variables feature strong interaction in the form of hierarchical relationships. Such hierarchy comes from activation relationships (variables determining whether other variables are active) and from value constraints (variables restricting the available options of other variables). Value constraints can be solved a priori (i.e. without running an evaluation), for example, by Satisfiability Modulo Theory (SMT) solvers. Due to the multidisciplinary nature of systems engineering, evaluation functions may include executing various discipline-specific physics-based analysis or simulation codes, which can be computationally expensive. Conflicting stakeholder needs, a common occurrence in systems engineering, leads to the presence of multiple conflicting minimization (or maximization) objectives. Design constraints are inequality constraints that cannot be solved for a priori, and therefore require an evaluation to be executed to steer the optimizer towards the feasible area in the design space. Such constraints may, for example, stem from physical (e.g. material stress limits) or operational limitations (e.g. space or energy usage). Finally, simulations used in the evaluation function might fail to converge: this is expressed in the optimization problem as a hidden constraint violation, which only provides the optimizer with the information that the constraint has been violated, but not by how much. For a more thorough discussion of the SAO problem challenges, the reader is referred to [4].

Model-Based Systems Engineering (MBSE) uses systems models to support the systems engineering process, however, traditionally these are only used to model individual system architectures [9]. Extensions of the Systems Modeling Language (SysML), such as CVL [10], VAMOS [11], or other approaches [5], allow modeling system variability (including architectural choices). Variability is an integral part of SysMLv2 [12], showing that it is considered an important capability to be supported in the future. An alternative method is using a feature models to generate architecture instances from 150% system models [13]. Other types of variability models include extensions of the Architecture Analysis and Design Language (AADL) [5], function-means models [14], and configurable components [15]. For a more in-depth overview of architecture design space modeling and optimization approaches, the interested reader is referred to [4, 5].

In this work, we use the Architecture Design Space Graph (ADSG) [4] to model the architecture design space and enable architecture generation. The ADSG is a directed graph modeling functions fulfillment choices, function induction by components, component characterization choices, and component connection choices. Nodes represent architecture elements such as functions, components, and component instances, and edges represent derivation relationships. Additional features include Quantity of Interest (QOI) nodes representing evaluation inputs (e.g. design variables) or outputs (e.g. performance metrics), connection choices, incompatibility constraints, and subsystem modeling. An ADSG model can be automatically translated into an optimization problem in terms of design variables, objectives, and design constraints. Value constraints are solved for by an algorithm presented in [4].

The ADSG is implemented in ADORE, a Python tool developed by the DLR that includes a web-based user interface for graphically modeling the architecture design space [16]. ADORE additionally contains interfaces to

various optimization algorithms and interfaces for connecting to the architecture evaluation code. Figure 1 shows the design space editing canvas. The ADORE model shows architectural choices by blue dashed arrows and defines several views: a system view showing functions, function derivation elements (components, decompositions, etc.) and ports, as well as a component view showing component-level elements such as attributes, component metrics and design variables, and port connectors.

Architecture evaluation code is problem specific and depending on what tools are useful for a given architecting problem can be implemented using different tools, environments, and/or programming languages. Generated architecture alternatives must be translated to the domain-specific input needed for the evaluation code. ADORE provides several different interfaces for this. For example, if evaluation is performed in Python code, ADORE's data model can be used to instantiate objects based on selected architecture elements. If a connection to an external environment is needed, generated architectures can be serialized to JSON or XML.

3 Space mission planning problem implementation

This section presents the application of SAO to space mission planning. The mission planning problem involves maximizing scientific value provided by a selected mission payload, subject to weight and power budgets. This application case was developed as part of the MBSE-Ops project [17], which had the goal of promoting MBSE adoption within the DLR.

3.1 Architecture generation: design space model and optimization problem definition

Architecture generation is enabled by modeling the architecture design space as an ADORE model, and using that model to formulate the optimization problem. Figure 2 shows the system view of the design space model, starting from the "Do Science" top-level function. This function represents the system-level goal and, therefore, also has the system-level performance parameters associated to it: "System Mass" and "Total Experiment Duration" act as optimization objectives, to be minimized and maximized, respectively. "Valid schedule" is a constraint that ensures the experiment schedule is valid (no experiments are overlapping in time).

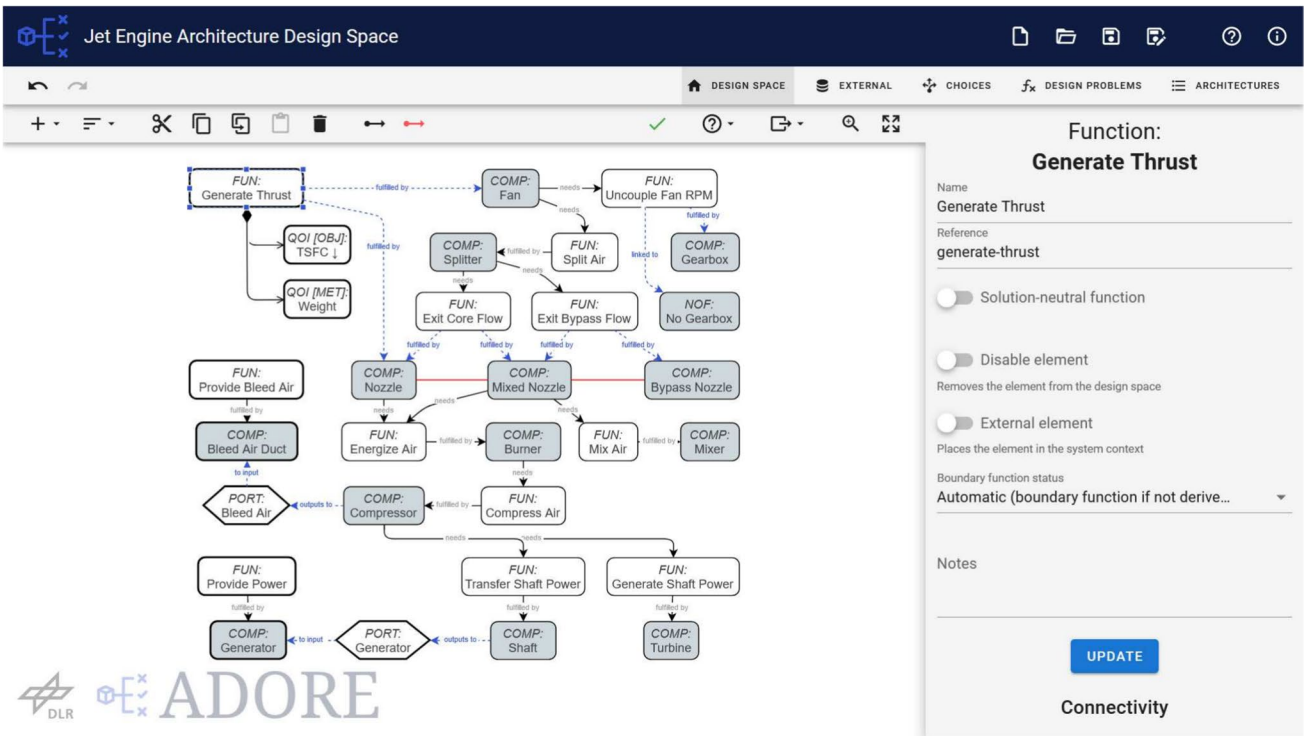

Fig. 1 ADORE user interface showing the design space editing canvas. Reproduced from [16]

The top-level function is fulfilled by the "Experiment" component, of which the detailed view is shown in Fig. 3. The Experiment can be instantiated between 1 and 5 times, and each instance has two continuous design variables: "Start Time" and "Duration". Instantiating the experiment and choosing values for the two design variables is a good example of decision hierarchy, a common occurrence in SAO problems, where an upstream decision (instantiation) determines whether downstream decisions (start time and duration) are active or not [4]. Note that the Experiment is not a physical component, but can still be considered an element of form as it represents a specific way to fulfill a function.

The experiment needs the "Gather Data" function, which is fulfilled by the "Instrument". The instrument contains no additional specialization, however, has two static inputs associated to it: "Mass" and "Power Requirement". Static inputs make assumptions and component properties explicit by moving their definition from the evaluation code to the system model. The Instrument needs two further functions: "Store Data", fulfilled by the "Onboard Computer", and "Provide Power", fulfilled by the "Battery". The Onboard Computer has "Mass" and "Power Requirement" static inputs associated with it, and additionally needs power, showing that it is possible for multiple components to need a function. The Battery, see Fig. 4 for the component view, can be instantiated between 1 and 5 times. Each instance has

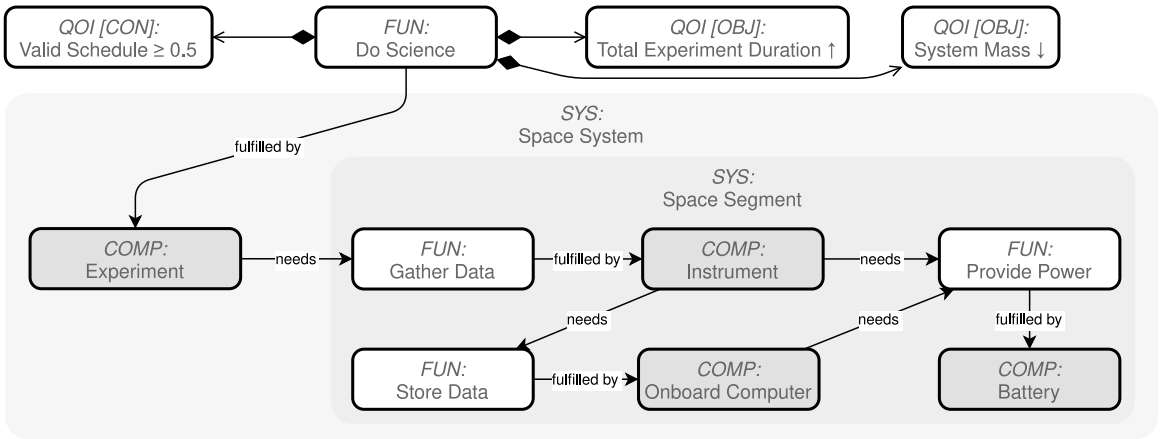

Fig. 2 ADORE design space model showing the system view; *QOI* Quantity of Interest, *FUN* Function, *COMP* Component, *SYS* System

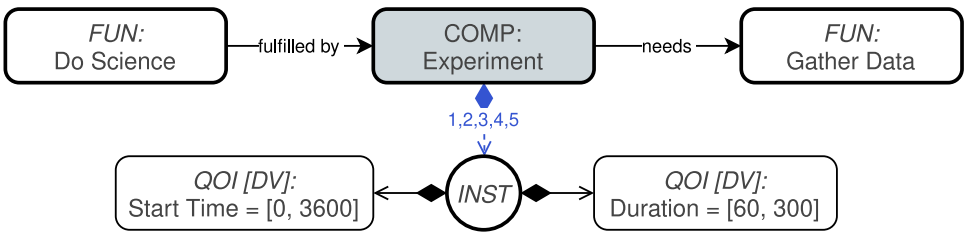

Fig. 3 ADORE design space model showing the "Experiment" component; *QOI* Quantity of Interest, *FUN* Function, *COMP* Component, *INST* Instance

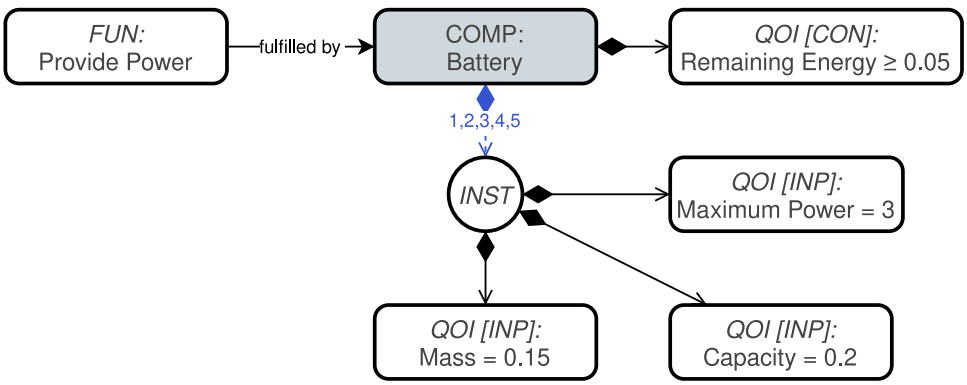

Fig. 4 ADORE design space model showing the "Battery" component. *QOI* Quantity of Interest, *FUN* Function, *COMP* Component, *INST* Instance

three static inputs: “Mass”, “Maximum Power”, and “Capacity”. On the component-level, the Battery has a “Remaining Energy” constraint that ensures that the battery is not depleted when running experiments.

The optimization problem is a mixed-discrete, constrained, multi-objective problem. It is defined by two discrete (instantiations) and ten continuous design variables (experiment start time and duration). It contains two objectives (System Mass and Total Experiment Duration) and two design constraints (Valid Schedule and Remaining Energy).

The purpose of the project presented in this work was to see if SAO can be applied to the design of space systems. Due to limited availability of domain experts, as well as time and resources limits, only a very simple application case could be implemented. That is why in this problem formulation, only the selection of the number of batteries influences system mass. In more realistic architecting problems, architectural choices related to other onboard equipment (including the instruments) would also influence the system mass.

3.2 Architecture evaluation: evaluation code and architecture translation

To implement the optimization problem, the analysis code needs to be flexible enough to enable calculating the same system-level performance metrics for all generated architectures. This is done by defining evaluation input in an object-oriented manner. By leveraging abstraction and inheritance, the implementation of some disciplinary analysis code is made easier. For example, for calculating total system mass, any component that has a mass can expose this through common class ancestry. Figure 5 shows the data model used as input performance calculation: the SpaceMission object contains the experiment schedule, represented by a list of Operation or Experiment classes, and one Spacecraft class that contains a list of Equipment classes. Several types of equipment are implemented, separated in PowerProvider and PowerConsumer classes. Currently, the only power provider is a Battery, and consumers are comprised of OnBoardComputer and Instrument.

This object-oriented structure of the data model brings several benefits:

- It provides an unambiguous interface between architecture generation and evaluation, enabling separation of concern: the evaluation code can be developed separately from the architecture design space model.
- It eases multidisciplinary calculation of system-level performance metrics by utilizing abstraction: for example, system mass can be calculated by simply summing all equipment masses, without regard to the actual equipment type.
- The data model can be extended independently from some of the calculations. For example, more types of power providers and consumers can be implemented without modifying code that analyzes power distribution.

The four performance metrics are calculated by the following disciplinary codes:

- The “mass analysis” discipline calculates the System Mass by summing all equipment masses.
- The “payload analysis” disciplines calculates the Total Experiment Duration by summing the duration of all operations of type Experiment.
- The “operations analysis” discipline determines whether the schedule is valid (the Valid Schedule flag) by checking if there are any overlapping operations.
- The “power analysis” discipline calculates the Remaining Energy by simulating power consumption over the mission duration, based on Battery capacities and required power of power consumers. Power consumers are either always on (e.g. the OnBoardComputer) or only consume power when used in an operation. Figure 6 shows power analysis output for an example mission containing two experiments.

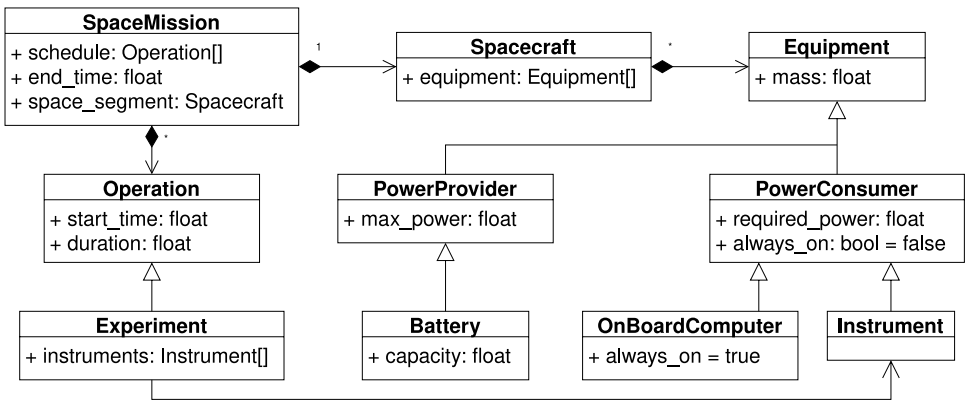

Fig. 5 Evaluation code data model

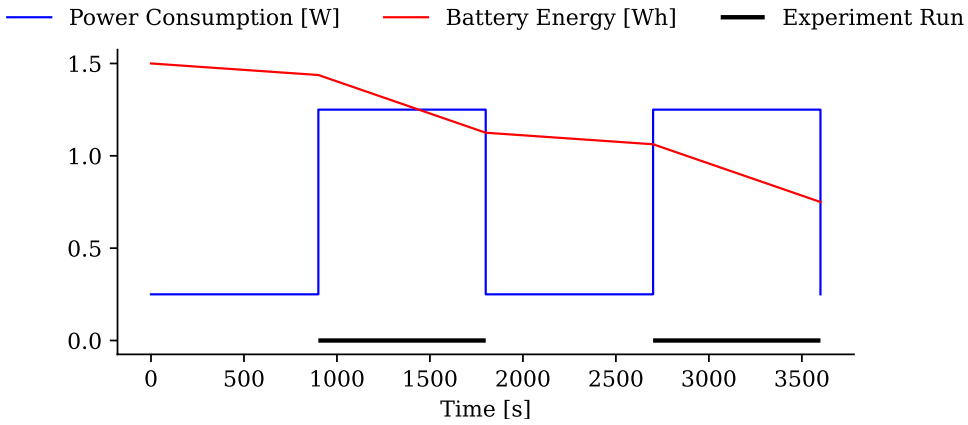

Fig. 6 Power analysis example for a mission architecture containing two experiments

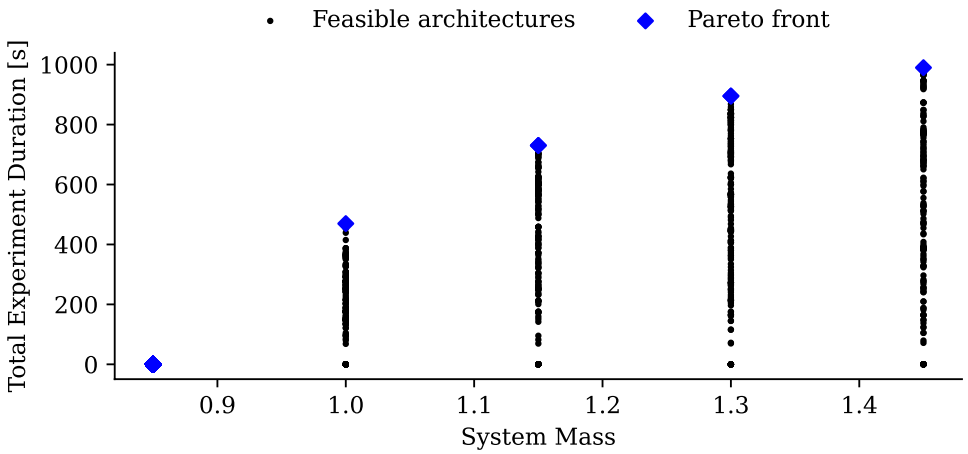

Fig. 7 Architecture optimization results, showing feasible architectures and architectures in the Pareto front

To enable architecture evaluation using ADORE, the architectures generated by ADORE have to be translated to the data model presented in Fig. 5. This is done using class factories: rules for instantiating Python classes based on architecture element occurrences. Class factories are defined for all objects in the design space, such that the top-level SpaceMission object can be populated. To illustrate, the class factory for the Experiment is defined as follows:

```
ClassFactory(
    # Associate this class factory to the "experiment" component
    el=ExternalComponentDef(name='experiment', auto=True),

    # Instantiate "Experiment" class if matched
    cls=Experiment,

    # Define class properties
    props={
        # Set start_time property to value of the Start Time QOI
        'start_time': ExternalQOIDef(name='start-time', auto=True),

        # Set duration property to value of the Duration QOI
        'duration': ExternalQOIDef(name='duration', auto=True),
    },
)
```

In this section, the architecture design space model, data model, and analysis disciplines have been presented sequentially, however, it has to be noted that all these aspects are strongly dependent on each other. The evaluation code is developed based on the elements and architectural choices modeled in the architecture design space, however, the design space can only include such elements and choices as to what is feasible to evaluate within a given project context. The data model enables coupling of analysis disciplines and coupling of the architecture generator to the evaluation code, and is usually developed over time by application in various projects [18].

4 Results

The architecture optimization problem defined in Sect. 3 is solved using NSGA-II [19], a multi-objective evolutionary algorithm that is well suited for solving mixed-discrete problems. The NSGA-II implementation in pymoo [20] is used, accessed through SBArchOpt [21]. SBArchOpt adds several interfaces on top of pymoo to aid solving architecture optimization problems. A design problem defined in ADORE can be directly exposed as an SBArchOpt problem, so no additional integration is needed from the user perspective. NSGA-II is executed with a population size of 100 for 20 generations.

Figure 7 presents the results of the architecture optimization. A Pareto front can clearly be observed, trading-off between system mass and experiment duration. The five (vertical) clusters are formed by the selection of the number of batteries, as this is the only architecture choice influencing mass in the application case. As expected, including more batteries increases mass, however, also increases energy availability and therefore allows more and/or longer experiments to be executed.

After identifying optimal candidate architectures, the next step is to inspect them and select one for implementation. Generated architectures can be inspected in the ADORE GUI. Even for the simplified design space used in the mission planning example, a resulting architecture can get quite complex. Figure 8 shows an automatic layout of such an architecture. Manual improvement of the architecture presentation is possible, but impractical if there is a number of candidate architectures or even several iterations of architecture optimization.

It is clear that better visualization and validation techniques would be required in productive applications. Within the context of the simplified example it was sufficient to validate the architecture and show the feasibility of the overall approach.

5 Conclusions

This work has demonstrated System Architecture Optimization (SAO) in the space domain using an exemplary space mission planning problem. The goal was to maximize science time while minimizing system mass. These two conflicting objectives resulted in a Pareto front of optimal architectures after executing the optimization. The architecture design space was modeled using ADORE, a tool for function-based architecture design space modeling and optimization problem execution. Architecture evaluation was implemented using a custom data structure consisting of Python classes, and four disciplines calculating four relevant performance metrics. Class factories were used to instantiate classes based on architecture element selection. This structure allows independent development of the architecture design space model, the evaluation data model, and disciplinary analysis codes.

In future studies, SAO may be applied to a more realistic space mission planning problems, involving more engineering disciplines and higher fidelity analysis. It can be expected that the tool landscape will become more heterogeneous if more disciplines are involved, potentially warranting a collaborative multidisciplinary optimization approach that is able to integrate distributed analysis tools into a single analysis workflow [22]. The architecture design space can be expanded to represent different types of experiments,

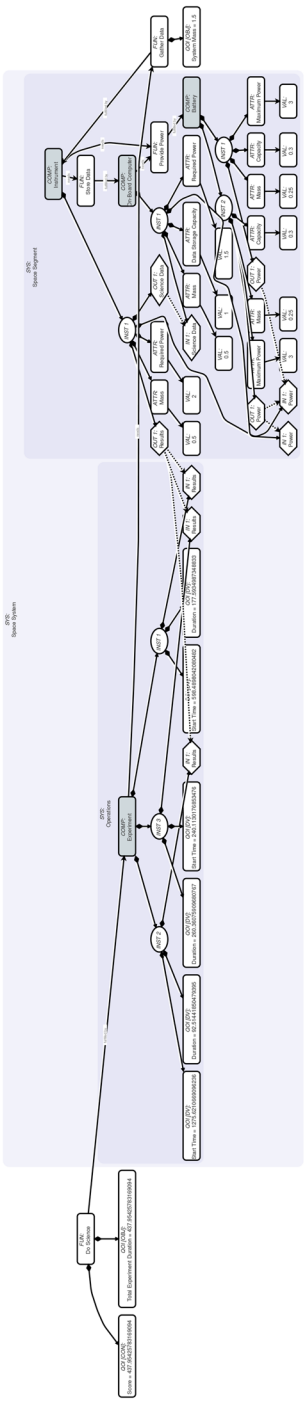

Fig. 8 Automatic layout of a resulting candidate architecture, showing already high complexity in the result for a small design space example

different types of instruments, and more realistic integration among on-board subsystems. The amount of programming required to setup an SAO problem should be reduced to make it easier to apply.

SAO should be integrated into established MBSE processes, interfacing with standard formats such as

SysML [23], Arcadia [24] or OPM [25], and it should be investigated how this would change the way of working for engineers. In the future, especially SysMLv2 [26] should provide an appropriate platform for integrating disciplinary analysis codes for engineering complex space systems using SAO.

Visualizing Pareto fronts with more than 2 dimensions remains a challenge due to the inherent limitations of representing high-dimensional data in 2D or 3D spaces. However, specialized plots such as scatter matrices and interactive data exploration tools can help mitigate these challenges and provide meaningful insights for decision-making processes. Visualization of the generated architectures should also be part of the decision-making process, and improvements are possible compared to current capabilities.

Acknowledgements This work was originally created in the frame of the MBSE-Ops project and presented at the MBSE2024 Workshop on Model-Based Space Systems and Software Engineering in Bremen, Germany on 28 May 2024, where it received the Best Paper Award in the category of scientific papers. The authors would like to thank the MBSE-Ops team for inspiring discussions and valuable feedback, and the MBSE2024 judging committee for the award, showing their interest in and support of the work.

Author contributions T.F. developed the space mission application case, including conceptualization, implementation, results generation and results processing. J.H.B. wrote the main manuscript text, T.F. prepared Figs. 6 and 7. Both authors reviewed the manuscript.

Funding Open Access funding enabled and organized by Projekt DEAL.

Data availability No datasets were generated or analysed during the current study.

Declarations

Conflict of interest The authors declare no competing interests.

References

1. MB4SE: MBSE best practices. Technical Report MB4SE-TN-002, ESA (2022)
2. McDermott, T.A., Folds, D.J., Hallo, L.: Addressing cognitive bias in systems engineering teams. In: 30th Annual INCOSE International Symposium, Virtual Event (2020). https://doi.org/10.1002/j.2334-5837.2020.00721.x
3. Fortescue, P., Swinerd, G., Stark, J.: Spacecraft Systems Engineering. Wiley, Hoboken (2011)
4. Bussemaker, J.H., Boggero, L., Nagel, B.: System architecture design space exploration: Integration with computational environments and efficient optimization. In: AIAA AVIATION 2024 FORUM, Las Vegas (2024). https://doi.org/10.2514/6.2024-4647
5. Procter, S., Wrage, L.: Guided architecture trade space exploration: fusing model-based engineering and design by shopping. Softw. Syst. Model. **20**(6), 2023–2045 (2021). https://doi.org/10.1007/s10270-021-00889-8
6. Crawley, E., Cameron, B., Selva, D.: System Architecture: Strategy and Product Development for Complex Systems. Pearson Education, England (2015)
7. Haberfellner, R., de Weck, O., Fricke, E., Vössner, S.: Systems Engineering. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-13431-0
8. Bussemaker, J.H., Saves, P., Bartoli, N., Lefebvre, T., Lafage, R.: System architecture optimization strategies: dealing with expensive hierarchical problems. J. Glob. Optim. (2024). https://doi.org/10.1007/s10898-024-01443-8
9. Menshenin, Y., Mordecai, Y., Crawley, E.F., Cameron, B.G.: Model-based system architecting and decision-making. In: Handbook of Model-Based Systems Engineering, pp. 1–42. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-27486-3_17-1
10. Broodney, H., Dotan, D., Greenberg, L., Masin, M.: Generic approach for systems design optimization in MBSE. INCOSE Int. Symp. **22**(1), 184–200 (2012). https://doi.org/10.1002/j.2334-5837.2012.tb01330.x
11. Weilkiens, T.: Variant Modeling with SysML. Leanpub, Victoria (2015)
12. Timperley, L., Berthoud, L., Snider, C., Tryfonas, T.: Mapping the MBSE environment and complementary design space exploration techniques. In: 2024 IEEE Aerospace Conference, pp. 1–20. IEEE, Big Sky (2024). https://doi.org/10.1109/aero58975.2024.10521188
13. Meinicke, J., Thüm, T., Schröter, R., Benduhn, F., Leich, T., Saake, G.: Mastering Software Variability with FeatureIDE. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61443-4
14. Gedell, S., Johannesson, H.: Design rationale and system description aspects in product platform design: focusing reuse in the design lifecycle phase. Concurr. Eng. **21**(1), 39–53 (2012). https://doi.org/10.1177/1063293x12469216
15. Raudberget, D.S., Edholm, P., Andersson, M.: Implementing the principles of set-based concurrent engineering in configurable component platforms. In: DS 71: Proceedings of NordDesign 2012, the 9th NordDesign Conference, Aarlborg University, Denmark (2012)
16. Bussemaker, J.H., Boggero, L., Ciampa, P.D.: From system architecting to system design and optimization: A link between MBSE and MDAO. In: 32nd Annual INCOSE International Symposium, Detroit, MI, USA (2022). https://doi.org/10.1002/iis2.12935
17. Boggero, L., Chojnacki, A., Bussemaker, J., Bartels, J., Quantius, D., Nagel, B.: The MBSE competence at the German Aerospace Center. INCOSE Int. Symp. **33**(1), 910–924 (2023). https://doi.org/10.1002/iis2.13061
18. Alder, M., Moerland, E., Jepsen, J., Nagel, B.: Recent advances in establishing a common language for aircraft design with CPACS. In: Aerospace Europe Conference (2020). https://elib.dlr.de/134341/1/AEC2020_174.pdf
19. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol.

Comput. **6**(2), 182–197 (2002). https://doi.org/10.1109/4235.996017
20. Blank, J., Deb, K.: Pymoo: multi-objective optimization in Python. IEEE Access **8**, 89497–89509 (2020). https://doi.org/10.1109/access.2020.2990567
21. Bussemaker, J.H.: SBArchOpt: surrogate-based architecture optimization. J. Open Source Softw. **8**(89), 5564 (2023). https://doi.org/10.21105/joss.05564
22. Ciampa, P.D., Nagel, B.: AGILE paradigm: the next generation of collaborative MDO for the development of aeronautical systems. Prog. Aerosp. Sci. (2020). https://doi.org/10.1016/j.paerosci.2020.100643
23. Friedenthal, S., Moore, A., Steiner, R.: A Practical Guide to SysML: The Systems Modeling Language, vol. 38. Elsevier, Amsterdam (2015). https://doi.org/10.1016/C2013-0-14457-1
24. Voirin, J. (ed.): Model-Based System and Architecture Engineering with the Arcadia Method. ISTE Press, London (2018)
25. Dori, D.: Model-Based Systems Engineering with OPM and SysML, pp. 1–411. Springer, New York (2016). https://doi.org/10.1007/978-1-4939-3295-5
26. Bajaj, M., Friedenthal, S., Seidewitz, E.: Systems modeling language (SysML v2) support for digital engineering. INSIGHT **25**(1), 19–24 (2022). https://doi.org/10.1002/inst.12367