
Enhancing Model-Based Systems Engineering (MBSE) Through Advanced Design Space Exploration Techniques

A Comprehensive Investigation

By

LOUIS RICHARD TIMPERLEY



Department of Aerospace Engineering
UNIVERSITY OF BRISTOL

A dissertation submitted to the University of
Bristol in accordance with the requirements of
the degree of DOCTOR OF PHILOSOPHY in the
Faculty of Engineering.

DECEMBER 2024

Word count: FILL IT IN

ABSTRACT

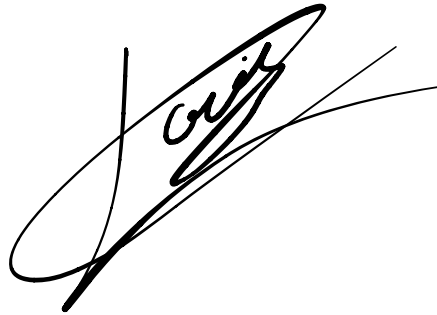
Here goes the abstract

DEDICATION AND ACKNOWLEDGEMENTS

Here goes the dedication.

AUTHOR'S DECLARATION

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

A handwritten signature in black ink, appearing to read 'Louis', with a large, sweeping flourish extending from the bottom left.

SIGNED: Louis Richard Timperley

DATE: August 14, 2024

CONTENTS

List of Tables	ix
List of Figures	xi
1 Introduction	1
1.1 Overivew	1
2 Literature Review	3
3 Research methods	5
4 Mapping the MBSE Environment and Complementary Design Space Exploration Techniques	7
4.1 Introduction	8
4.2 Variability Framework	9
4.3 Review	9
4.4 Design Space Exploration Techniques	31
4.5 MBSE Knowledge Graph	35
4.6 Discussion and Future Work	40
4.7 Conclusion	43
5 Developing a Data-Driven Ontology for Design Variability of Earth Observation Spacecraft	45
5.1 Introduction	46
5.2 Method Definition	53
6 Demonstrating Variability Framework Guided System Parameter Op- timization of Earth Observation Spacecraft	63

CONTENTS

6.1	Introduction	64
6.2	Method	66
7	Space Architecture Generation using LLMs	75
8	Discussion	77
9	Conclusion and Future Directions	79
A	Appendix	81

LIST OF TABLES

4.1	Design variability framework	9
4.1	Common MBSE Languages	11
4.3	SysML V1 Extension for Trade-Studies	15
4.5	Common MBSE Tools	18
4.7	Common MBSE Methods	21
4.9	Common Simulation Tools	25
4.11	Identified design space exploration issues	28
4.13	Rules used for issue severity	31
4.1	Candidate design space exploration techniques	32
4.1	Example MBSE Context	40
5.1	ISO 15288 System Architecting mapping to example MBSE methods, [89], [50]	49
5.3	Types available in the Architecture Design Space Graph (ADSG) and their related variability concepts, see [96] for further detail	51
5.5	Types available in the Model Based Product Line Engineering (MBPLE) framework and their related variability concepts, see [96] for further detail . .	52
5.1	Query Types and details	60

LIST OF FIGURES

4.1	Knowledge base ontology and MBSE process model	37
4.2	MBSE environment type definitions in the knowledge graph	38
4.3	Flowchart showing the steps used to identify MBSE processes with the knowl- edge graph	39
4.4	Example MBSE process identified using the search algorithm, process starts on the left and ends on the right	41
5.1	Relative Cost to Fix Software Errors per Life Cycle Phase [86]	47
5.2	Overview of the framework elements and its context	53
5.1	System architecting process, making use of a domain knowledge base	54
5.2	Static ontology for system architecting	57
5.3	Examples of domain specific design rules	58
6.1	Steps taken when using the toolset	69
6.2	Steps undertaken for each design space exploration	72

CHAPTER 1

INTRODUCTION

I have passed through fire and deep water, since we parted. I have forgotten much that I thought I knew, and learned again much that I had forgotten.

— J.R.R. Tolkien, *The Lord of the Rings*

1.1 Overview

Model Based Systems Engineering (MBSE) replaces the documents used by traditional system engineering approaches with a unified and coherent model, covering multiple viewpoints of the system. In doing so, it removes inconsistencies in design information commonly injected during Document Based Systems Engineering (DBSE) activities, where design changes would require updates in multiple documents. While MBSE has been shown to successfully describe a system baseline, its applicability could be extended greatly widened by introducing tools for rapid exploration of the design space. The primary language used by MBSE, SysML, has limited native support for describing how elements of a design may change from one design option to another [1], and many common MBSE methodologies do not address trade-offs or iterating of the design in detail. MBSE has been lacking in this area as until now most projects have relied on manual iteration and exploration of designs. This however involves a large quantity of

information travelling across various different interfaces in the project, including different disciplines and tools. Delays and errors in this information flow can drive up design time and cost. The PhD aim is to enhance MBSE through advanced design space exploration techniques. It will therefore aim to answer the following (provisional) research questions:

RQ1A: What are the limitations of current MBSE tooling and processes for use in design space exploration, in the spacecraft domain? (world as it is now)

RQ1B: What are the advanced design space exploration methods that offer potential benefits to MBSE? (world as it is now)

RQ2: How does the MBSE tooling and processes need to be changed or expanded to incorporate design space exploration techniques? (intervention on the world)

RQ3: What are the benefits brought to MBSE and the spacecraft domain by exploring different designs and what are the limitations? (impact on the world)

While the knowledge contributions this PhD will provide are expected to be generalisable to many engineering domains, spacecraft will be taken as the primary focus for application in this work. Spacecraft systems are an ideal application for MBSE. Their complexity, and corresponding volume of documentation, mean that it is challenging to keep documents up to date, but any error poses a very high risk. As a result, the potential benefits of using MBSE are of particular value when applied to this domain. Progress so far, including the first two years, has covered initial literature review, MBSE tools/languages/methods review, identification of research aim /questions, two case studies of specific design space exploration techniques and a technique-agnostic design variability framework. The rest of this report is split up as follows. Section 2 covers a set of identified design variability levels, Section 3 presents a candidate thesis structure, section 4 covers the planned steps to completion, section 5 discusses the technical work and publications completed so far and section 6 gives conclusions.

LITERATURE REVIEW

Dust is only a name for what happens when matter begins to understand itself. Matter loves matter. It seeks to know more about itself, and Dust is formed.

— Phillip Pullman, *The Amber Spyglass*

RESEARCH METHODS

Dust is only a name for what happens when matter begins to understand itself. Matter loves matter. It seeks to know more about itself, and Dust is formed.

— Phillip Pullman, *The Amber Spyglass*

MAPPING THE MBSE ENVIRONMENT AND COMPLEMENTARY DESIGN SPACE EXPLORATION TECHNIQUES

Any work of science, no matter what its point of departure, cannot become fully convincing until it crosses the boundary between the theoretical and the experimental: Experimentation must give way to argument, and argument must have recourse to experimentation.

—Gaston Bachelard, *The New Scientific Spirit*

Today's MBSE tools and environments are highly varied and therefore present a challenge for organizations looking to implement MBSE. Furthermore, while MBSE environments are highly capable of supporting the description of design baselines, the current capabilities within these environments could be further refined for exploring alternative designs. As a result it is important to gain an understanding of the limitations of current MBSE tooling in performing the valuable activity of design space exploration, and identify a set of candidate techniques to combat these. This paper reviews the various options available to MBSE practitioners by comparing some of the most common MBSE languages, tools and methods. The possible issues that can be encountered when exploring different designs have been identified and assigned a severity rating. A set of design

space exploration techniques are presented, and where possible these have been sourced from existing literature. A knowledge graph has been constructed to collect all this data into a structured format, containing all the MBSE languages, tools, methods, design space exploration-related issues and techniques, as well as the relationships between each of these. This knowledge graph, implemented as a Neo4j graph database, allowed deeper insights to be drawn from the collected information. By defining a selected MBSE environment, including language, tool and method, the knowledge graph could be used to identify the least troublesome sequence (with minimum number of related issues) to arrive at a desired design artifact, for example a set of optimized system parameters. Beside this, the knowledge graph could be used to display the relationships and clusters of MBSE languages, tools and methods, to assist organizations with selecting suitable MBSE environment elements. Future work will bring greater depth to the analysis available with the knowledge graph, for instance, differentiation between different types of design space exploration issues and techniques.

4.1 Introduction

Model Based Systems Engineering (MBSE) is a maturing method, the tools used to setup a capable and successful MBSE environment are still evolving. As a result, it can be difficult to select the language, tool, method and support tools that are suitable for an organization's or project's goals and context. Further to this, methods for exploiting MBSE's unique benefits when performing design space exploration are not yet fully understood. Some work in this area has already been performed [1],[2],[3],[4],[5] however there are still many untested candidate techniques for design space exploration with MBSE and further investigation is necessary to fully justify wide adoption in industry. The goal of this study was to firstly review and map the primary elements of today's MBSE environments in a way that would be useful to new practitioners setting up their own environments. Secondly this study reviews the short-comings in these MBSE environments when exploring alternative designs. With reference to a theoretical framework for design variability discussed in the next section, these issues are mapped to candidate techniques for possible solutions, some of which have already been applied in literature and others as yet untested. The rest of this paper is split as follows; section 2 presents the variability framework used as part of this study, section 3 describes the findings of the review, section 4 presents the capture of the review findings in the form of a knowledge

Table 4.1: Design variability framework

Variability Level	Description	Example
Parameter	Numbers, e.g., mass	Mass, Orbit Altitude
Multiplicity	Number of elements in a group	Number of Redundant Reaction Wheels
Topological	Configuration of interfaces between parts	Centralised topology vs. federated
Element type	Type of parts used in the system	Solar array or radio-isotope generator

graph, section 5 discussion and future work and finally section 6 gives conclusions.

4.2 Variability Framework

To enable a more precise categorisation of how designs can vary between alternatives, a variability framework was established. Shown in Table 4.1, this framework has been used to categorise the capabilities of different design space techniques and more precisely understand where issues lie in current MBSE practice.

It is expected that the most frequently sought level of variability in design problems is ‘parameter’. Element type level is likely the second most important. Topological is largely constrained by the specific components included in a design, and multiplicity can in many cases be approximated by parameter level variability in a model. This framework does not cover aspects such as variation in exploration objectives or requirements themselves. Further work is planned to refine the framework, discussed later.

4.3 Review

A literature review has been carried out, covering a wide range of options available for MBSE environments, as well as issues with current practice and candidate design space exploration techniques. This review considered MBSE environments to be comprised of three elements [6]:

- Language - The way that information is captured, used to specify all the various aspects of the system [7].

- Tool - The software or programme used to create the model.
- Method - The series of steps followed to populate the model using the language in the tool.

Any formal model will follow an ontology, or a description of a set of concepts and their inter-relations [8], this will specify how the abstract language is used to describe the real system. Though it focuses on the language, it tends to be specific to particular methods, therefore ontologies have been considered part of the method for this review.

4.3.1 Languages

Table 4.1 gives a summary of the range of modelling languages covered in the review. Green cells are positive, yellow are non-ideal, red is negative.

Table 4.1: Common MBSE Languages

Language	Developer	Tool	Method	Variability Modelling	Simulation Capability	Customisation
UML	OMG	Flexible	Flexible	Generalization and multiplicity across relationships	Good (though can be slow)	Can be extended via profiles
SysML V1	OMG	Flexible	Flexible	Similar to UML plus parametric diagrams may also be used to support alternative analysis	Good (though can be slow)	Can be extended via profiles

Language	Developer	Tool	Method	Variability Modelling	Simulation Capability	Customisation
SysML V2	OMG (Future release)	Flexible	Flexible	Product line engi- neering and variability points	Good	Can be extended via exten- sible model libraries
Capella	Eclipse	Capella	ARCADIA	None in- built	Limited (on third party tools)	Open Source
Object-Process methodology	OPM/ISO	Flexible	OPM	Limited/none	Animation	Limited
System Defini- tion Language (SDL)	Vitech	GENSYS	Vitech MBSE Methodol- ogy	Limited/none	Behaviour simulation in GENSYS	Is extensi- ble
Valispace lan- guage	Valispace	Valispace	Valispace	Limited/none	Parametric	Web based, has API
RePoSyD Lan- guage	RePoSyD	RePoSyD	RePoSyD	Limited/none	Limited/none	Open source

Language	Developer	Tool	Method	Variability Modelling	Simulation Capability	Customisation
AADL	SAE	Flexible	Flexible	Limited/none	Can be simulated, e.g. in TASTE for model base verification	Can be extended via language annexes
SLIM	COMPASS	COMPASS	COMPASS	Limited/none	Performability analysis and others, focussed on safety and correctness	Limited

UML

UML is a diagram-based language, designed with software development in mind. Its objective is to allow engineers to analyse, design, and implement software systems as well as modelling of business and similar processes [9]. There are several diagrams available, such as class diagrams, composite structure diagrams, package diagrams, activity diagrams, state machine diagrams and sequence diagrams among others. It has only limited provision for describing design variability, whereby generalization relationships can be defined between elements. This enables specific variants to inherit properties from a “general” element type, that may then be further refined. Multiplicity may also be specified for association, aggregation and composition relationships. While suitable for modelling product lines with well defined and finite differences (for example a range of car models with varying features), these methods become cumbersome to use when exploring design alternatives not explicitly modelled by the user.

SysML V1

Has been widely used across MBSE tools and applications in industry, and has become somewhat synonymous with the concept of MBSE [10]. Developed from UML, it has good coverage of systems engineering concepts, but with similarly limited variant modelling capability (using the same generalization and relationship multiplicity methods). However the SysML spec does include an extension for trade studies shown in Table 4.3 [11] Other extensions can be built to provide new types for the language via profiles, that are sets of custom “stereotypes”, meaning custom types specific to the users needs and domain.

SysML V2

An upcoming version of SysML V1, with many major changes hoped to address the shortcomings identified with the original [12]. Firstly, Version 2 has a more layered approach compared to V1. These layers are; System syntax, Kernel syntax (this has its own specification), Core syntax and Root syntax. Besides this, it will add a textual version of the language, that is intended to be equally expressive. For better integration with ontology tools and languages, it will have integration with the OWL/RDL ontology languages. This allows it to have good mapping to data ontologies such as ESA’s Overall

Table 4.3: SysML V1 Extension for Trade-Studies

Stereotype	Base class	Description
Objective Function	Constraint Block	An objective function is used to determine the overall value of an alternative in terms of weighted criteria and/or MOE's (measure of effectiveness)
MOE	Property	A measure of effectiveness (MOE) represents a parameter whose value is critical for achieving the desired mission cost effectiveness

Semantic Modelling for System Engineering (OSMoSE)[13]. On top of product line engineering capabilities similar to UML and version 1, a major benefit of version 2 will be more support for variability modelling, better enabling trade-offs and alternative configurations as well as decision points [14].

Capella Language

The Capella language (used by the Capella tool) is similar to SysML but with some notable differences. Capella functions have no control flow description (meaning conditions for particular sequences of functions) as Arcadia (the Capella method) is purely focused on architectural design and interface definition [15]. There is no Capella requirement diagram, but requirements can be shown in any diagram in Capella. No parametric diagram exists in Capella, but can generally handle the concepts it requires in other ways. Finally, the Capella language has no variant modelling WAIT SEE <https://mbse-capella.org/resources/arcadia-reference/Arcadia>

Object Process methodology language

First developed in 1995, the Object Process Methodology (OPM) uses a conceptual modelling language for capturing knowledge and designing systems [16]. In a similar fashion to SysML V2, OPM has two versions of the language. Firstly object-process diagrams, or the diagram-based version of the language and the object-process language, or the text based version of the language. OPM is focussed on modelling both objects and processes including the links between them [17]. Both objects and processes are

modelled as OPM “things” and links are either modelled as structural or procedural [18]. Unlike UML/SysML that follows a multi-model structure (with behaviour spread through 13 types of diagram) OPM is a single model structure. This helps with understanding of the model and avoids the model-multiplicity problem, where concurrent understanding of multiple sub models is required [19].

Some examples of using the OPM language to model variants exist in literature, but with no application to systems engineering [20].

System definition Language

The System Definition Language (SDL) is used as part of the Vitech MBSE Methodology, STRATA. SDL is based on the following concepts [21]:

- SDL Entities are similar to nouns in English and refer to objects
- Relationships are similar to verbs and refer to a link between two entities
- Attribute are further descriptions of entities in the same way adjectives modify nouns
- Adverbs also have a corresponding element in SDL with Attributed-Relationships (i.e., attributes on relationships)

Valispace language

The language used by the Valispace tool is highly data driven [22] and so largely focusses on describing parameters and their relationships.

RePoSyD Language

This is another language tied to its method and tool. Its definition is based on a Meta model, essentially a description of allowed elements and relationships in models and similar to an ontology. Meta models may also be found in UML/SysML models.

The Architecture Analysis and Design Language

The Architecture Analysis and Design Language (AADL) was originally designed for architecting software and hardware systems for avionics [23]. It facilitates direct code generation and can be used as a form of documentation itself.

SLIM

Based on AADL, SLIM is another tool-tied language for the COMPASS toolset [24].

4.3.2 MBSE Tools

Table 4.5 gives a summary of the modelling tools covered in the review. Green cells are positive, yellow are non ideal, red is negative.

Cameo Systems Modeler (No Magic)

Cameo Systems Modeler (CSM) is a widely used MBSE tool aimed for use with UML/SysML V1 (it does not enforce a particular method). Now called Magic Systems of Systems Architect, it is a closed source programme but can be customized and extended via plugins. CSM includes powerful simulation capabilities via the Cameo Simulation Toolkit that allows direct execution of SysML models. Further to this it includes integrations with MATLAB, python and provides an open API. All of this capability has ensured CSM is widely used in industry, for example by JPL [25] and Airbus [26]. The integrations offered by CSM could be leveraged to support analysis of alternatives, for example as done in [27].

Capella

Capella [28] is an open-source tool for use with the Capella language and Architecture Analysis and Design Integrated Approach (ARCADIA) method. Its simulation capabilities are smaller compared to CSM, but it is able to simulate state machines and functional flows. MATLAB can be integrated with the tool, but generally simulation is heavily reliant on third party tools. Capella is the standard tool used by ESA and Thales Alenia Space, who were both involved with its development.

Papyrus

An open source SysML based tool. Much like CSM it has no specific related method, however unlike CSM it has extremely limited simulation capability, making bottom-up style analysis difficult.

Table 4.5: Common MBSE Tools

Tool	Developer	Language	Methodology	Simulation	Customisation
Cameo	Dassault	SysML V1	Flexible	Good (though can be slow)	Closed source, Open API, custom plug-ins
Capella	Thales	Capella	ARCADIA	Limited (reliant on third party tools)	Open Source
Papyrus	Eclipse	SysML V1	Flexible	None	Open Source
Valispace	Valispace	Valispace	Valispace	Parametric	Web based, Closed source
Rhapsody	IBM	SysML V1/UML	Flexible	Good	Closed source, Open API, custom plug-ins
Enterprise Architect	Sparx	SysML V1/UML	Flexible	Limited to simulation of behavioural diagrams	Closed source
OPM Cloud	OPMCloud	OPM	OPM	Limited animation style simulation, some optimisation tools inbuilt	Web based, closed source
GENSYS	GENSYS	(Ecosystem of tools, SDL for MBSE)	STRATA	System behavioural simulation	Closed source
RePoSyD	RePoSyD	RePoSyD	RePoSyD	Limited	Open source
TASTE	ESA	SDL, AADL	Flexible	System behavioural simulation and testing for validation	Open Source
COMPASS 3.0	COMPASS/ESA	SLIM	COMPASS 18	Performability analysis and others, focussed on safety and correctness	CLI interface and some open-source aspects

Valispace

An example of web-based MBSE tool is Valispace. While it may be more difficult to customise (being web based), it does offer an API for further integrations. As previously discussed, it is highly oriented towards capturing engineering data, rather than being concerned by specific semantics, leading to it sometimes being referred to as a tool for Data-Driven Systems Engineering (DDSE).

IBM Rhapsody

IBM Rhapsody is a further UML/SysML tool that has been in use since 1996 and has reached a high level of maturity. It has simulation capabilities built in “for design-level debugging” [29].

Sparx Systems Enterprise Architect

Another UML/SysML modelling tool is Sparx Systems Enterprise Architect, capable of simulation of behavioural diagrams such as State machines, interaction (sequence) diagrams, activity diagrams etc. It can generate executable code for state machines for application and simulation and supports Hardware Description Languages (HDL, such as Ada, VHDL and Verilog).

OPMCloud

OPMCloud is the web-based tool that supports the object process method language. Unlike most other tools, it incorporates means to optimise the system model, based on pareto front, Design-Structure Matrix (DSM)-based methods, or even graph database queries [30].

GENSYS

GENSYS is a manufacturing-focussed MBSE tool, covering capabilities such as; product data management, production planning and control and shop floor control

RePoSyD

The RePoSyD tool is tied to the RePoSyD method/ language

TASTE

TASTE is a freely available toolset for development of embedded real-time systems [31], developed by ESA and various partners. It primarily focusses on modelling of embedded systems and can generate code from external modelling languages, such as SDL or Simulink, as well as supporting typical embedded languages such as C++, VHL, ADA etc. It has other typical MBSE tool capabilities built in such as validation of models [32].

COMPASS

COMPASS (Correctness, Modelling and Performance of Aerospace Systems) 3.0 a toolset providing an integrated model based approach for system software co engineering in the aerospace domain [33]. It uses the SLIM language and includes various analysis capabilities [34] such as; functional correctness checking, Safety and dependability analysis, performability analysis, fault detection identification and recovery effectiveness analysis and finally requirement validation.

There is an ongoing effort to combine TASTE and COMPASS, to form COMPASTA [35].

4.3.3 MBSE Methods

Table 4.7 summaries the MBSE Methods covered in the review. Green cells are positive, yellow are non ideal, red is negative.

ARCADIA

The Architecture Analysis and Design Integrated Approach (ARCADIA) is the method used by the Capella language/tool. It follows 5 stages all intended to produce a system architecture and flow down requirements to the component level [36].

- Define the Problem – Customer Operational Need Analysis,
- Formalisation of System/SW Requirements – System/SW Need Analysis,
- Development of System/SW Architecture – Logical Architecture,
- Development of System/SW Architecture – Physical Architecture,

Table 4.7: Common MBSE Methods

Method	Developer	Language	Tool	Design Space Exploration
ARCADIA	Eclipse/ Thales	Capella	Capella	Limited
OOSEM	Lockheed Martin and SSCI	SysML V1	Flexible	Limited
Harmony	IBM, Hoff- man, Dou- glass	SysML V1	Rhapsody	Supported within the design synthesis stage
RePoSyD	Hoppe	RML	RePoSyD	Limited
STRATA	Vitech	SDL	GENSYS	Limited
OPM	Dori	OPM	OPMcloud	Limited
Valispace	Valispace	Valispace	Valispace	Limited
MOFLT	Airbus	SysML V1	Flexible	Limited, but has tailor- ing in mind
COMPASS	COMPASS	COMPASS	COMPASS	Limited
SEAM	University of Bristol	SysML V1	Flexible	Limited

- Formalize Components Requirements – Contracts for Development and IVVQ (Installation, Verification, and Validation Qualification)

It has very little design space exploration provision besides one example in literature using a third party tool [37].

OOSEM

The Object-Oriented Systems Engineering Method is a top down, scenario-driven process for SysML. It supports analysis, specification, design and verification of systems and is generally applied recursively at each level of the system [38].

Harmony

Intended to be used with IBM's Rhapsody, the method covers the system development from requirement analysis to system acceptance [39]. The main steps in the process are:

- Requirement Analysis – Collection and completion of requirements and categorisation as functional or non-functional. Requirements are allocated to use cases. The models at this stage are not executable
- System Functional Analysis - The use cases are analysed and executable use case models are created and verified against requirements
- Design Synthesis – This is the stage where most trade-offs and design exploration may be performed. Behaviour from the use case models are merged into a single system model. Functionality is allocated to subsystems, allowing the formation of an architectural concept model, including each subsystem. The resulting executable model is then verified against requirements.
- SW Analysis and Design - The model is now handed off to SW and HW development. The specifics of this stage is not covered explicitly by the harmony process.
- SW implementation and unit Test - The implementation of SW and subsystems is not covered explicitly by the harmony process.
- Module Integration and Test, followed by (sub-) System integration and test – the various SW and HW modules are integrated and verified against requirements, followed by the subsystems and then full system.
- System Acceptance – Finally, if the full system is successfully verified against requirements, it may be accepted, finishing the process.

RePoSyD

The Requirements Engineering, Project Management, and System Design (RePoSyD) method, developed in 2017 [40], is based on the IEEE-1220 systems engineering process definition [41]. It has its own cloud based tool and language (RML) and includes some provision for basic trade-offs[42].

STRATA

STRATA, provided by Vitech, uses the SDL language and covers four main domains [17]:

- Requirements - starting from top level system requires, these are further refined through the process [43].

- **Functional/Behaviour** - Description of what the system must do to achieve its mission. Top level functions are decomposed as the process advances [44].
- **Architecture/Synthesis** - Physical Architecture of the system, where physical components are assigned functions, ensuring the physical design is separated from the logical design and avoid the premature allocation of behaviour to physical components [43].
- **Verification And Validation** - Each physical component and behaviour is verified against relevant requirements already included in the model.

Object Process Method

The object process method, used by OPMCloud, uses two main mechanics. Firstly in-zooming is a refinement, typically of a process, that produces a timeline of events occurring in that process. Unfolding is a process that analyses the system structure, identifying objects in the system. Dynamic and Structural aspects can be represented in the model. Dynamic aspects modelled by relationships between processes and objects in an Object-Process diagram (OPD), and structural represented by “relations between two things of the same class (Process-to-Process) or (Object-to-Object) within an OPD” [17]. There is no dedicated diagram for requirements, instead they are modelled as a state of an object and a “satisfies/is implemented by/derives” link is added to relate the requirement to its associated elements. [45]

Valispace

The Valispace method is intended to address many aspects of systems engineering, such as requirement engineering, detailed design, simulation, test, review, technical change management and documentation. The main focus of the whole language, tool and method ecosystem is to facilitate concurrent design and the early phases [46]. Concurrent design “is a systematic approach to integrated product development that emphasises the response to customer expectations. It embodies team values of co-operation, trust and sharing in such a manner that decision making is by consensus, involving all perspectives in parallel, from the beginning of the product life-cycle” [47]. It has seen wide application across the spacecraft industry and, amongst others, ESA has become a significant

supporter of this way of running early phase design studies, setting up a dedicated Concurrent Design Facility (CDF) [47].

MOFLT

The MOFLT method (Mission Operations Functional Logical Technical), developed at Airbus Defence and Space, is language and tool agnostic. Compliant with ECSS and with tailoring points for extensibility, the method has 5 layers [48]; mission, operational architecture, functional architecture, logical architecture, technical/physical architecture. The mission layer covered mission definition and determining of potential mission realisations. The operational analysis layer covers operational concept definition and operational scenarios. The functional architecture includes functions, sequence of functional execution and structural arrangement of functions. The logical architecture defines the system logical components and allocates functions to each. Finally, the physical architecture is the physical (or real) implementation of the system. There are strong similarities between MOFLT and ARCADIA, with the primary difference being that the functional architecture is given its own layer, instead of being spread across the system analysis, logical and physical architecture layers [49].

SEAM

The Spacecraft Early Analysis Model framework (SEAM), was developed at the University of Bristol, in partnership with Airbus Defence and Space [50]. Much like ARCADIA and MOFLT, SEAM is made up of five packages, or sub models, each capturing a different aspect of the system, Mission, Operation Logical architecture, Functional architecture and Simulation. Firstly, the Mission package contains information about the system purpose, requirements, external entities and external interfaces. The operations package describes what actions the system shall perform while it is operating. The two architecture packages are closely intertwined and reference each other at many levels of the system. The logical architecture defines the constituent parts of the system (logical elements), such as subsystems, assemblies and components, while the functional architecture describes the functions available to the system and what logical elements perform them. The simulation package contains information about the simulations, for example the mission phases to be stimulated and start and stop mission times. SEAM is targeted

Table 4.9: Common Simulation Tools

Tool	Developer	Language	Interfacing/ model genera- tion	Customisation
Modelica	Modelica	Modelica	Common (e.g. cameo)	Open Source
Simulink	MathWorks	Simulink/ MAT- LAB	Common (e.g. cameo)	Closed Source
Collimator	Collimator	Collimator (strong coupling with Python)	Less Com- mon	Closed (limited API) Source python
Cameo Simu- lation Toolkit	NoMagic/ Das- sault Systems	Sys/ML V1;UML	Directly from SysML or UML	Open API
Rapsody	IBM	Sys/ML V1;UML	Directly from SysML or UML	Open API

for use with Cameo Systems Modeler and SysML, therefore the simulations package can make use of the Cameo Simulation Toolkit for direct simulation of the system model.

4.3.4 System Simulation

Table 4.9 presents an overview of the dedicated simulation tools considered for this review. Green cells are positive, yellow are non-ideal, red is negative.

Modelica

Modelica is an object-orientated modelling language, with integration for many tools including cameo systems modeller. It has the potential to allow a dual model approach [51] with a high level system model, for example in SysML, for requirements and Modelica model for simulation. Open Modelica is an example of free Modelica environment [52].

Simulink

Simulink provides 7 primary capabilities [53], namely, intuitive description of system architectures, domain specific tools and prebuilt blocks for modelling multi-domain systems, integration of multiple team's tools into one simulation, simulation and analysis of system behaviour, parallel simulation for batch analysis and finally deployment of simulation as standalone applications. Further to this, Simulink has integration support for many tools, but as the developer, MathWorks, does not provide an API (and Simulink is closed source), these integrations can be somewhat unreliable.

Collimator

Collimator [54] is a closed source, web based, data driven systems modeller with similar diagramming to Simulink. Besides simulation and analysis, it can be used to generate C and Python code. It has built in version control and a python API for loading models and running simulations. Furthermore, Collimator can import data, in CSV, JSON, Text and XML formats for creating series, for example, describing input parameter variation through time.

Cameo Simulation Toolkit

Cameo simulation toolkit [55] is built into Cameo Systems Modeller (now Magic Systems of Systems Architect) and is capable of full behavioural simulation and parametric modelling, allowing execution of all relevant SysML/UML diagram types. The toolkit has a capable API for accessing simulation elements.

Rhapsody

Similarly to cameo simulation toolkit, Rhapsody's built-in simulation capabilities allow animated execution of all relevant SysML/UML diagram types, alongside an API for interfacing to external tools. Unlike cameo simulation toolkit, simulations in Rhapsody do not have a simulation clock, however it is still possible to model asynchronous behaviour and some language constructs such as state machine joins and forks not available in cameo simulation toolkit are available in Rhapsody.

4.3.5 Domain Issues

With all of the above information collected, it was possible to identify a set of issues, or shortcomings of the current state of MBSE practice related to design space exploration. These issues are listed in Table 4.11. The list is by no means exhaustive but includes the most immediate issues. Note that the rules in Table 4.13 were used to identify the issue severities.

Table 4.11: Identified design space exploration issues

Name	Summary	Affected Aspects	Severity
Parameter variation language	The language lacks convenient ways to describe how a design's parameters change between different design options	UML, SysML V1, Capella Language, OPM Language, SDL, Valispace Language, RML, AADL, SLIM	5
Topology variation language	The language lacks convenient ways to describe how a design's topology changes between different design options	UML, SysML V1, Capella Language, OPM Language, SDL, Valispace Language, RML, AADL, SLIM	4
Multiplicity variation language	The language lacks convenient ways to describe how a design elements' multiplicity changes between different design options	UML, SysML V1, Capella Language, OPM Language, SDL, Valispace Language, RML, AADL, SLIM	3
Element variation language	The language lacks convenient ways to describe how a design elements' type changes between different design options	UML, SysML V1, Capella Language, OPM Language, SDL, Valispace Language, RML, AADL, SLIM	5
Tool based variation	The tool lacks convenient ways to switch between different design options	Cameo, Capella, Papyrus, Valispace, Rapso- dy, Enterprise Architect, OPMCloud, GENSYS, RePoSyD Tool, TASTE, COM- PASS 3.0	4

Name	Summary	Affected Aspects	Severity
Exploration stage in methodology	There is no well-defined stage at which exploration of design alternatives (in physical architecture) should take place	ARCADIA, OOSEM, RePoSyD, STRATA, OPM, Valispace Method, MOFLT, COMPASS, SEAM, TASTE Method	3
Identification of what aspects to vary and at what level	There is no well-defined method for selecting aspects of a design to vary (apart from starting from nothing)	ARCADIA, OOSEM, RePoSyD, STRATA, OPM, Valispace Method, MOFLT, COMPASS, SEAM, TASTE Method	3
Simulation time	Simulations can become expensive to run for many design alternatives, especially if the analysis model used cannot be simplified	Cameo, Capella, Papyrus, Valispace,hapsody, Enterprise Architect, OPMCloud, GENSYS, RePoSyD Tool, TASTE, COMPASS 3.0	4
Selection of designs	The designer cannot select the “best” design among many without rigorous metrics	ARCADIA, OOSEM, RePoSyD, STRATA, OPM, Valispace Method, MOFLT, COMPASS, SEAM, TASTE Method, UML, SysML V1, Capella Language, OPM Language, SDL, Valispace Language, RML, AADL, SLIM	2

Name	Summary	Affected Aspects	Severity
Synthesis of new alternatives	Efficiently numerating design aspects between alternative designs is difficult	Cameo, Capella, Papyrus, Valispace, Rhapsody, Enterprise Architect, OPMCloud, GENSYS, RePoSyD Tool, TASTE, COMPASS 3.0	2
Synthesis of Novel alternatives	It is difficult to encourage an “algorithmic” process to incorporate the inventiveness of a human designer in synthesising new design options	Cameo, Capella, Papyrus, Valispace, Rhapsody, Enterprise Architect, OPMCloud, GENSYS, RePoSyD Tool, TASTE, COMPASS 3.0	2
Design Space Coverage	Guaranteeing that the design space has been fully explored is a highly open ended and complex task	ARCADIA, OOSEM, RePoSyD, STRATA, OPM, Valispace Method, MOFLT, COMPASS, SEAM, TASTE Method	4
Constraints on feasible designs	It is not possible to describe what designs are feasible, or compliant with requirements	UML, SysML V1, Capella Language, OPM Language, SDL, Valispace Language, RML, AADL, SLIM	3

Table 4.13: Rules used for issue severity

Condition	Severity
Issue prohibits any form of design exploration	5
Issue prohibits most forms of design exploration	4
Issue prohibits a few forms of design exploration or presents a challenge to carry out the exploration process	3
Issue limits the quality of the design exploration process in some way	2
Issue has only a minor impact on some aspect of the design exploration process	1

4.4 Design Space Exploration Techniques

With the identification of current MBSE practice covering languages, tool and methods, alongside design space exploration related issues, it was possible to start exploring candidate techniques to integrate into current practice that would address its short comings. The exploration techniques included both methods of generating new design alternatives, and the evaluation of these alternatives. Various domains were considered, not necessarily related to MBSE, with the expectation that many techniques used in other domains could be adapted. The necessary inputs and outputs as well as applicable types of design space (type of variability) vary for each technique. For example some may require simulation or some form of system analysis, and some are only capable of varying continuous parameters while others can be adapted to various types of variability. The selection of techniques should be driven by firstly what kind of design exploration is being done, in reference to the variability framework presented in Table 4.1, what inputs are available and what are the desired outputs. Each technique, required input, output and variability type is recorded in Table 4.1.

Table 4.1: Candidate design space exploration techniques

Name	Summary	Variability type/ Issues Solved	Inputs	Outputs	Advantages	Disadvantages
Surrogate modelling [56]	Mathematical approximation model of full parent model, useful when it is expensive to simulate the parent model	Parameter/ Simulation Time	Training points from parent model	Mathematical approximations of parent mode	Significantly reduce simulation time of complex models	Needs training, not robust to discontinuities
Combinatorial optimisation [57]	Discrete type of optimisation-find an optimum set from a finite set of objects (e.g. travelling sales man problem)	Discrete/ Selection of designs	Finite set of possible system element variants, (element type variation or discretised parameters etc.)	Optimum set of system element types/discretized parameters	Relatively simple to implement	Can be NP complete (Non-deterministic Polynomial complete) leading to long simulation time
Gradient based optimisation [58]	Follows direction of steepest descent to find minimum	Parameter/ Selection of designs	System model, objectives	Local/global Minimum	Faster convergence on minima	If not agent based, can get trapped in local minimum
Genetic optimisation [59], [60], [61]	Uses repeated cross over and mutation between a population of individual solutions to find minimum	Parameter/ Selection of designs	System model, objectives	(typically) Global minimum	Less susceptible to getting stuck in local minima, Tuneable mutation and crossover	Computationally expensive
Particle swarm optimisation [62]	A similarly bio inspired method, much like genetic optimisation, that considers a set of particles that behaves in a similar way to a flock of birds in order to find the minimum of a design space	Parameter/ Selection of designs	System model, objectives	(typically) Global minimum	Less susceptible to getting stuck in local minima	Computationally expensive
Simulated annealing optimisation [63]	Models the process of heating then cooling a material to minimise the model's total energy as an analogue for finding the minimum of a design space	Parameter/ Selection of designs	System model, objectives	(typically) Global minimum	Less susceptible to getting stuck in local minima	Computationally expensive

Name	Summary	Variability type/ Issues Solved	Inputs	Outputs	Advantages	Disadvantages
Interactive Optimisation [64]	A human “expert” is able to make inputs to the optimisation process, either by changing search directions or parameters etc.	Dependant on user interaction/ Selection of designs	System model, objectives, Expert “intuition”	(typically) Global minimum, and selection of “unmeasured” aspects performed	Can better be fused with “expert” intuition	Requires manual input
Rule based design generation [65]	Discrete rules for if such and such an aspect of the organisation or requirements are present, then include such and such an element in the design	All/ Synthesis of new alternatives	Organisational info, mission requirements	System architecture	Rapidly arrive at solution	Limited to rule domain
Design of experiments [66]	Statistical process for identifying configuration of tests with the aim of identifying what key factors influence outputs, what setting should they be in acceptable outputs, main interactions in the process, how to minimise variation in the outputs	Parameter/ Design Space Coverage, Identification of what aspects to vary and at what level	System model, objectives, simulation results	Test points	Maximise usefulness of training/sample points	Need to run more simulations to identify sensitivities
Evolutionary generative design [67]	Similarly to genetic optimisation, using a stochastic evolutionary algorithm to generate designs (ideally from scratch), but relying on the designer to perform the selection of the best option	All/ Synthesis of new alternatives, Synthesis of novel alternatives	Requirements, Expert “intuition”	Set of varied designs	Possibility of diverse set of designs suggested to designer/engineer	May become challenging for engineer to make decision on best option
Deep Reinforcement learning for design concept generation assistant [68]	Select a component selection from a database (action) then evaluate this (e.g. by using a parametric model). Calculate a reward for this, then use this to train on a neural network, the action-reward mapping	All/ Synthesis of new alternatives	Parametric system model, database of components	Product breakdown (component list)	Identify optimum component selection, generally faster than combinatorial approaches	Must first train Neural net to learn reward-action mapping, limited to component database scope

Name	Summary	Variability type/ Issues Solved	Inputs	Outputs	Advantages	Disadvantages
SysML V1/UML Generalization Relationships [69]	By defining a “general” element and linking each design alternative with this general element using a generalization relationship, properties can be inherited and refined for each design alternative	Discrete/Parameter, multiplicity and element type variation language	Finite set of possible system element variants, (element type variation or discretised parameters etc.)	System architecture, including a set of design alternatives (per element, not at system level)	Relies only existing language constructs in UML and SysML V1	Can only include variants explicitly modelled by the user, cumbersome to explore these variants at the systems level
Parametric Variability SysML V1 Stereotypes [27]	New SysML V1 stereotypes for describing the parameter level variation of design variables	Parameter/Parameter variation language	Logical Architecture, Functional Architecture, Physical Architecture	Parameter Variability	Enables parameter level variability description in SysML models	limited to parameter level
Objective SysML Stereotypes [27]	New SysML stereotypes for describing the objectives related to the model	Parameter/Description of a good design	Logical Architecture, Functional Architecture, Physical Architecture	Objectives	Enables a simple selection method of designs, by which a better value of objectives means a better design	Can become ambiguous with many conflicting objectives
Design constraint and dependant variable SysML Stereotypes [27]	New SysML stereotypes for describing dependant variables and constraints related to them to identify compliant/feasible and infeasible/noncompliant designs	Parameter/Description of constraints on feasible designs	Logical Architecture, Functional Architecture, Physical Architecture	Design constraints	Enables description of where the feasible design region is	difficult to describe more complex constraints than a parameter being within a certain range
Swarm intelligence (SI) and multi-agent societies [70]	Commonly used to model social or collective behaviours, they are capable of autonomous action using their individual priorities. Essentially a set of unsophisticated individuals interact local with their environment and cause a system or global level functional pattern - emergent behaviour	All/ Synthesis of new alternatives, Synthesis of novel alternatives	Requirements, Parametric System Model	System architecture	“Regular design evaluation and improvement, Multiple solutions, Optimisation, Disruptive innovation” [70]	heavy computation requirement “Progress slows down after achieving near optima solutions” [70]

4.5 MBSE Knowledge Graph

To be able to gain further insight into the data gathered during the review, it was decided to model the information in a knowledge base. This would structure the information in a more machine readable way than simple tables and text. A knowledge base describes a set of real world things and the relationships between them, thus is essentially synonymous with the concept of ontology [71]. It should be noted that knowledge bases/ontologies can contain both conceptual definitions of types or classes, and instances (the population of the ontology). The knowledge base followed the basic ontology/schema defined in Figure 4.1. A Neo4j graph database [72] was used to implement this knowledge graph. Neo4j was picked because of its flexibility for data modelling, access to built-in network analysis tools, and its powerful query language called Cypher [73]. [74] presents an approach for using Neo4j in support of system analysis in conjunction with MBSE.

4.5.1 Querying and Reasoning Using the Knowledge Graph

Fig.2 gives an overview of the entire knowledge graph in Neo4j (provided simply for illustrative purposes). Various new insights could be drawn from the graph using Neo4j's query language, Cypher [73]. For example, the following cypher fragment would return the complementary design space exploration techniques associated with the SEAM method (linked by common artifacts):

```
$ MATCH (technique : Technique) <- [:TAKES_AS_INPUT] - (artifact :
    Artifact) <- [:GENERATES] - (method : Method {uid : 'SEAM'})
$ RETURN technique
```

Similarly for tools, in particular Cameo Systems Modeller:

```
$ MATCH (technique : Technique) <- [:TAKES_AS_INPUT] - (artifact :
    Artifact) <- [:GENERATES] - (method : Method) <- [:CAN_IMPLEMENT] - (
    tool : Tool {uid : 'Cameo'})
$ RETURN technique
```

For languages, such as SysML V1:

```
$ MATCH (technique : Technique) <- [:TAKES_AS_INPUT] - (artifact :
    Artifact) <- [:GENERATES] - (method : Method) <- [:CAN_IMPLEMENT] - (
    tool : Tool) <- [:AVAILABLE_IN] - (language : Language)
```

\$ **RETURN** technique

Further to this, simple reasoning could be performed, meaning the derivation of new information from the graph. Specifically by following the ontology in Figure 4.1 it was possible to infer node categories, such as languages, tools and methods all being examples of MBSE environment element, without this needing to be explicitly stored in the knowledge base. This could be performed using the following cypher fragment:

```
$ CALL n10s.inference.nodesLabelled('MBSE_Environment_Element',{
$ catNameProp: "dbLabel",
$ catLabel: "Type",
$ subCatRel: "NARROWER_THAN"})
$ YIELD node
$ RETURN node.uid as uid, labels(node) as categories
```

The CALL n10s.inference.nodesLabelled clause calls an inference method that adds the node label MBSE Environment Element to nodes that are of a type defined as NARROWER THAN the supertype MBSE Environment Element (meaning that the narrower than relationship exists between them). See Figure 4.2. While this is a simple inference to make, future work is hoped to identify further subtypes to each of the MBSE environment elements, issues and techniques, making this inference capability more useful.

With this reasoning capability the knowledge base was extended to a full knowledge graph, following the definition; a “knowledge graph acquires and integrates information into an ontology and applies a reasoner to derive new knowledge” [75]. Reasoning is a process by which some new piece of information is inferred by a knowledge graph [76], performed by some “reasoner” mechanism.

4.5.2 Network Analysis

When undertaking a project using MBSE, the process was modelled starting with the selection of a language, then tool, then method, that in turn generates a set of system engineering artifacts, after which design space exploration techniques may be employed taking certain artifacts as inputs and generating new ones as outputs. The overall MBSE process finishes once a particular artifact has been created, for example, detailed system design. This is shown in fig.1. This allowed different MBSE processes to be explored

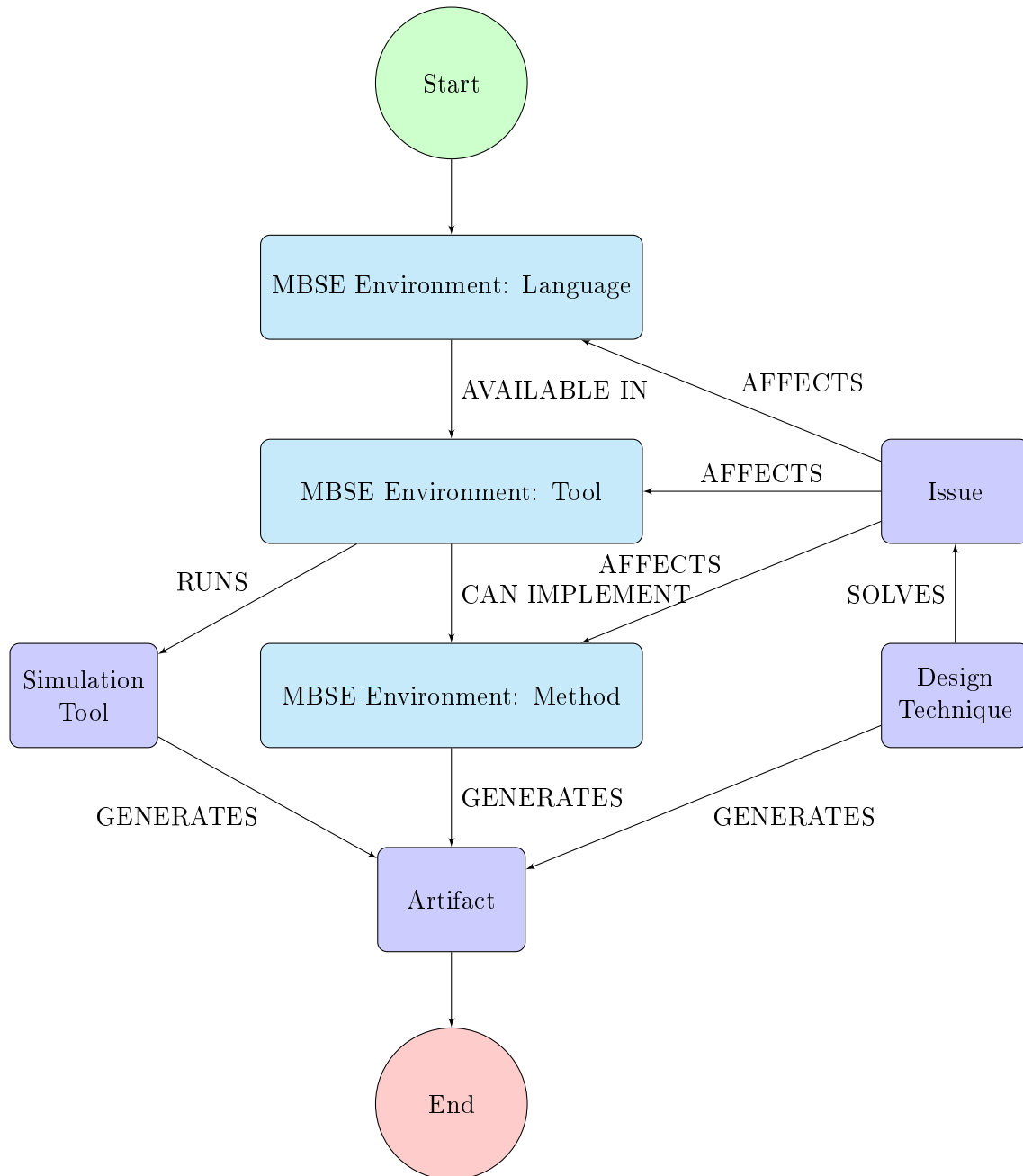


Figure 4.1: Knowledge base ontology and MBSE process model

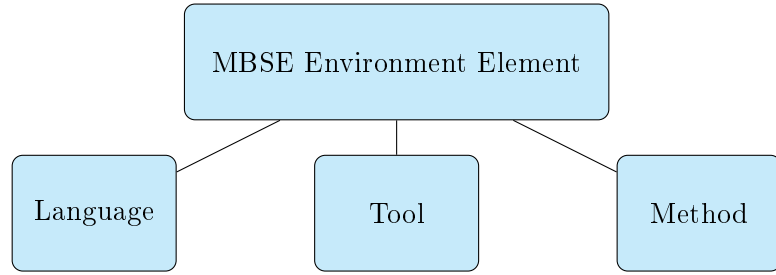


Figure 4.2: MBSE environment type definitions in the knowledge graph

and the number and significance of related issues compared using the knowledge graph. By adding different design space exploration techniques to the process, certain issues could be “solved”, therefore their issue cost as part of an MBSE process would be set to zero. For example, using surrogate modelling as a technique would address the issue of simulation time, see Table 4.11 and Table 4.1. The details of how each issue would be solved varies, and is not explicitly modelled in the knowledge graph, beyond the removal of the specific issue cost when a relevant technique is used.

A search algorithm was developed for identifying efficient MBSE processes using the knowledge graph, leveraging network analysis tools offered by Neo4j. Dijkstra’s algorithm [77] was used to identify the shortest path between nodes, thus the most efficient process or process with the lowest issue cost. This algorithm was chosen over alternatives such as the A* shortest path algorithm [78] (that uses a heuristic based approach to guide traversal of nodes) as it was simple to implement and the size of the graph being considered was relatively small.

Issue cost, the proxy for path length used for Dijkstra’s algorithm, was computed as the sum of a node’s related issue severity. For example the SEAM method that has 4 issues related to it in the knowledge graph, had a total issue cost of 12, counting each issue’s severity.

Some logic for identifying pre-requisites for certain parts of a process were added. For example methods creating particular artifacts needed for selected design space exploration techniques. The concept of a “context” was also created, that included optional descriptions of selected language, tool, method, simulation tool and desired destination. Design space exploration techniques could be “preselected” as part of this context, in which case the algorithm would fill in the gaps, by suggesting any additional techniques required to reach the desired destination artifact, or the algorithm could be left to select

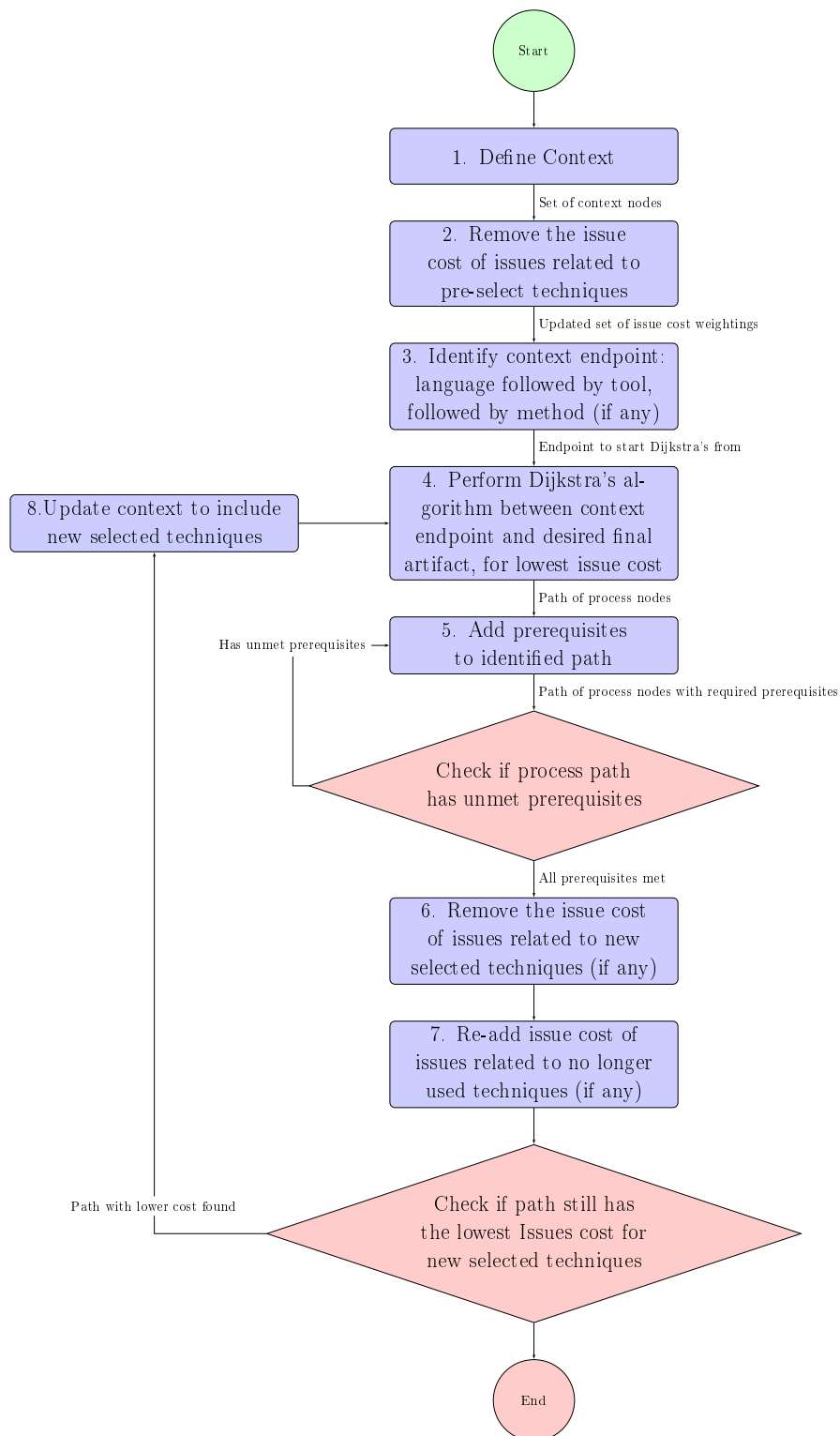


Figure 4.3: Flowchart showing the steps used to identify MBSE processes with the knowledge graph

Table 4.1: Example MBSE Context

Context Element	Selection
Language	SysML V1
Tool	Cameo Systems Modeler
Method	SEAM
Simulation Tool	Cameo Simulation Toolkit
Pre-Selected Design Space Exploration Techniques	Constrained Genetic Optimisation, Surrogate Modelling
Desired Destination	Globally Optimal Design Parameters

all of the techniques on its own. As techniques selected as part of the process would “resolve” issues, this led to an update in path issue costs. As a result, an outer loop needed to be added to the Dijkstra’s algorithm and prerequisite identification steps to ensure that the process path identified still had the lowest possible issue cost. Therefore the search algorithm could iterate on its selected techniques and eventually converge on the most efficient MBSE process for the given context. This process followed by the search algorithm is presented in Figure 4.3.

The flexibility in information provided in the search algorithm input meant that it could be performed with a range of constraints to suit an engineer working on a project in an organisation that might have access to only certain languages/tools/methods/techniques, and still identify efficient MBSE processes that are relevant to the project context.

???? shows an example of a path generated using this method, with the context shown in Table 4.1. The path identified includes the techniques; Objective SysML stereotypes (either using measures of effectiveness already included in SysML V1 or the new stereotypes proposed in [27]), parametric SysML Stereotypes [27], design constraint and dependant variable stereotypes and finally constrained genetic optimisation.

4.6 Discussion and Future Work

The currently available MBSE environments generally fall into two categories; those based on a specialised MBSE language and those using SysML V1 (or in future V2).

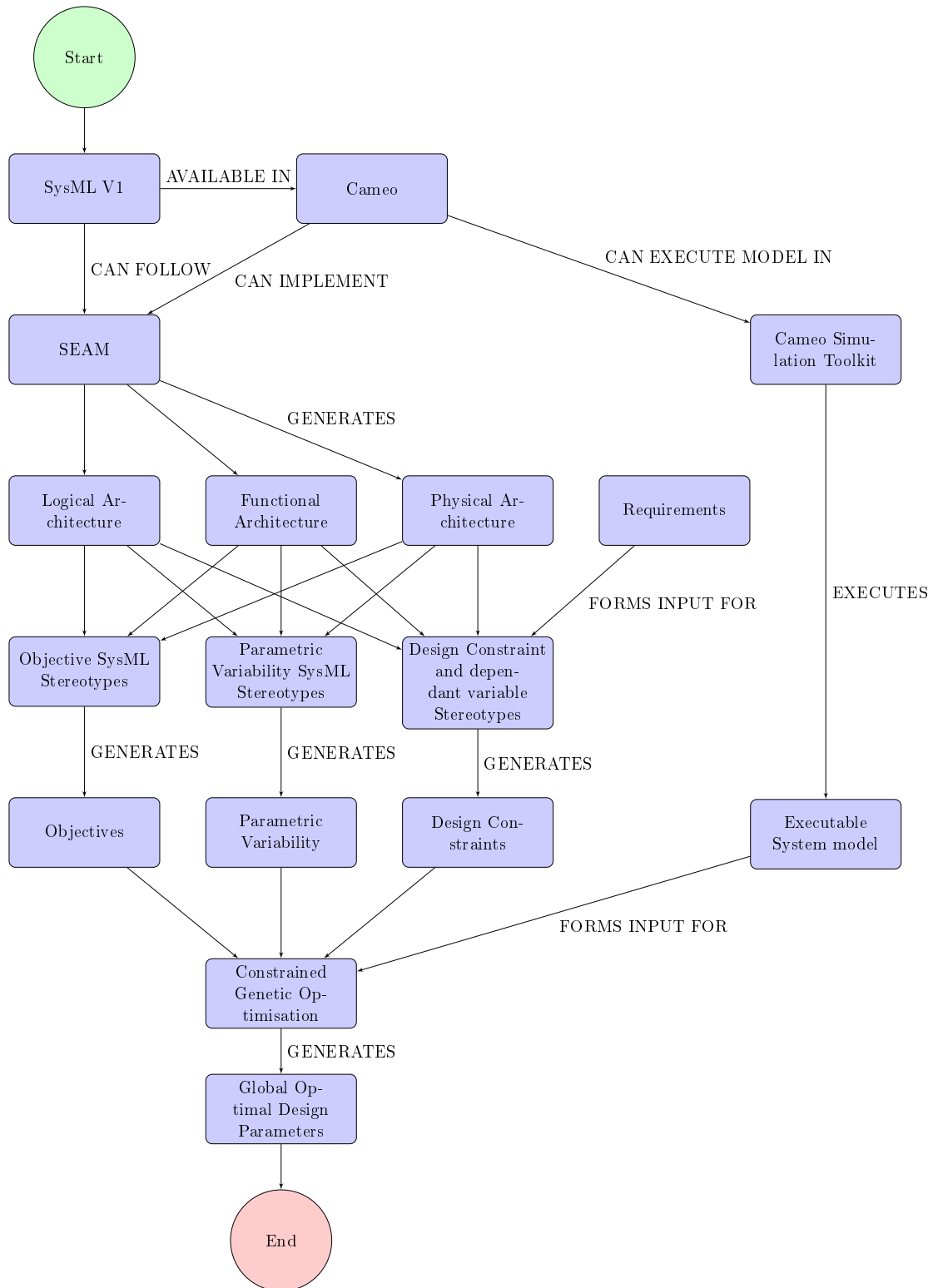


Figure 4.4: Example MBSE process identified using the search algorithm, process starts on the left and ends on the right

SysML is the most widely used MBSE language, with many tools and methods developed with it in mind. This, in part, is because of its flexibility, allowing practitioners to tailor the language to better fit their needs. Besides this, the fact that the language is a descendant of UML has likely also helped it to gain traction. Some of the tools and methods that are not related to SysML are focussed on a particular domain, such as COMPASS, with its focus on system-software co-engineering [34], and therefore see more specialisation in capabilities, such as COMPASS's generation of FMEA tables and FDIR assessment features.

SysML V2, while still in the early stages of release, may help alleviate some of the issues raised here, particularly by providing basic design space description capabilities. However, these new features will need to be linked to techniques that can identify and select new designs.

Using the completed knowledge graph presented in section 4, it was possible to consider a wide range of candidate techniques for design space exploration with MBSE and as new ones are identified, these can easily be entered into the knowledge graph and their usefulness measured. The knowledge graph however is still only an approximation of the true problem. As a result, the design of case studies assessing new techniques will still rely on judgement of factors not covered by the knowledge graph, such as access to related expertise and time required to implement versus execute.

The issues presented cover a wide range of aspects of MBSE elements and required capabilities for design space exploration. These could be further categorized alongside the proposed design space exploration techniques, which may assist in identifying issue solutions. For example issues related to the description of the design space (or variant modelling), as opposed to say selection of a particular design in that space, would have clear links to techniques that provide these design space descriptions.

The variability framework used for this study could be expanded to cover the potential for varying the objectives of the design space exploration, and further detail of the current state of the system design (beyond simply the MBSE environment used to produce it). Besides this, provision for sensitivity of different designs would be a useful inclusion, as well as further refined variation subtypes. Work beyond the scope of this paper is currently under way to address these shortcomings.

Finally, the languages, tools and methods presented here do not represent an exhaustive list, and clearly further additions to the knowledge graph will benefit its usefulness.

4.7 Conclusion

This study has reviewed a number of today's MBSE languages, tools and methods. The interrelations between these MBSE environment elements have been mapped and a set of related issues and shortcomings for exploring different system designs has been identified. Design space exploration techniques have been proposed as candidate solutions to these issues and all the information collected during the review has been collated into a knowledge graph to allow further understanding and analysis of potential MBSE processes. The code and data used to populate and interact with the knowledge base maybe found in a GitHub repository [79].

CHAPTER 5

DEVELOPING A DATA-DRIVEN ONTOLOGY FOR DESIGN VARIABILITY OF EARTH OBSERVATION SPACECRAFT

Model Based Systems Engineering (MBSE) is an interesting alternative to traditional systems engineering methods. Instead of using electronic documents to record system information, MBSE uses a unified and coherent system model. Trade-offs are a major element of a space systems engineer's role in early system design. This can be a particularly challenging process in the domain of spacecraft, as the system designs are often very complex and the constraints can be difficult to characterize. There has been little previous research on the use of MBSE as a design space exploration tool or in support of trade-offs. This paper investigates the potential to use MBSE for design exploration and to understand trade-offs, through the creation of a new toolset including a SysML profile. The tool draws on generative design (allowing automatic guided generation of a multitude of design alternatives) and system optimization to rapidly generate and assess new designs using interactive analysis and visualizations. Techniques such as surrogate modelling, genetic algorithms and robustness measurements will be available in the toolset. The toolset was applied to a design scenario aiming to improve the trade off and design selection process for LEO Earth observation satellites. The upcoming ESA TRUTHS space mission was used as a case study and the design process was recorded and com-

pared to a manual design exploration approach. The toolset was found to reduce the design exploration time by 38% to 96% , allow exploration of more designs in an equivalent time and provide better quantification of the relationships present in the design space, all without drops in selected design quality. For now, the toolset can only perform parameter variation in the design exploration and future work is expected to extend this to higher levels of variability. The study also discusses how the MBSE toolset could be applied to other missions, offering the same advantages to all early phase spacecraft designers.

5.1 Introduction

In order to understand the benefit of flexible domain knowledge exploitation in the context of system architecting, it is first necessary to explore the current practice in the area.

5.1.1 The System Architecting Process

The process of system architecting is the first step that develops a system's requirements and objectives into high level descriptions of the system itself [80]. Many options for a system's architecture may be considered during this stage of development, however the selected architecture should have features, properties and characteristics that satisfy the initial set of requirements [81]. This linkage back to requirements is called traceability and means that every element in the architecture has a justification for its inclusion in the system, found in the requirements [82]. Furthermore, this places the focus of the system architecting process on the system's suitability to meet requirements. This is somewhat different to the later stages of development where design processes focus on the system's compatibility with technologies, feasibility of construction and system integration [83]. The importance of the system architecting process in supporting a successful system development cycle is difficult to over state. It is where the problem or opportunity the system is addressing is fully understood and where its most fundamental elements are defined. The decisions made here therefore carry the most influence and will have consequences affecting the rest of the system life cycle. "All the really important mistakes are made the first day" [84]. Another way to consider this is that the cost of correcting a design error typically rises exponentially through the system's lifecycle [84], [85].

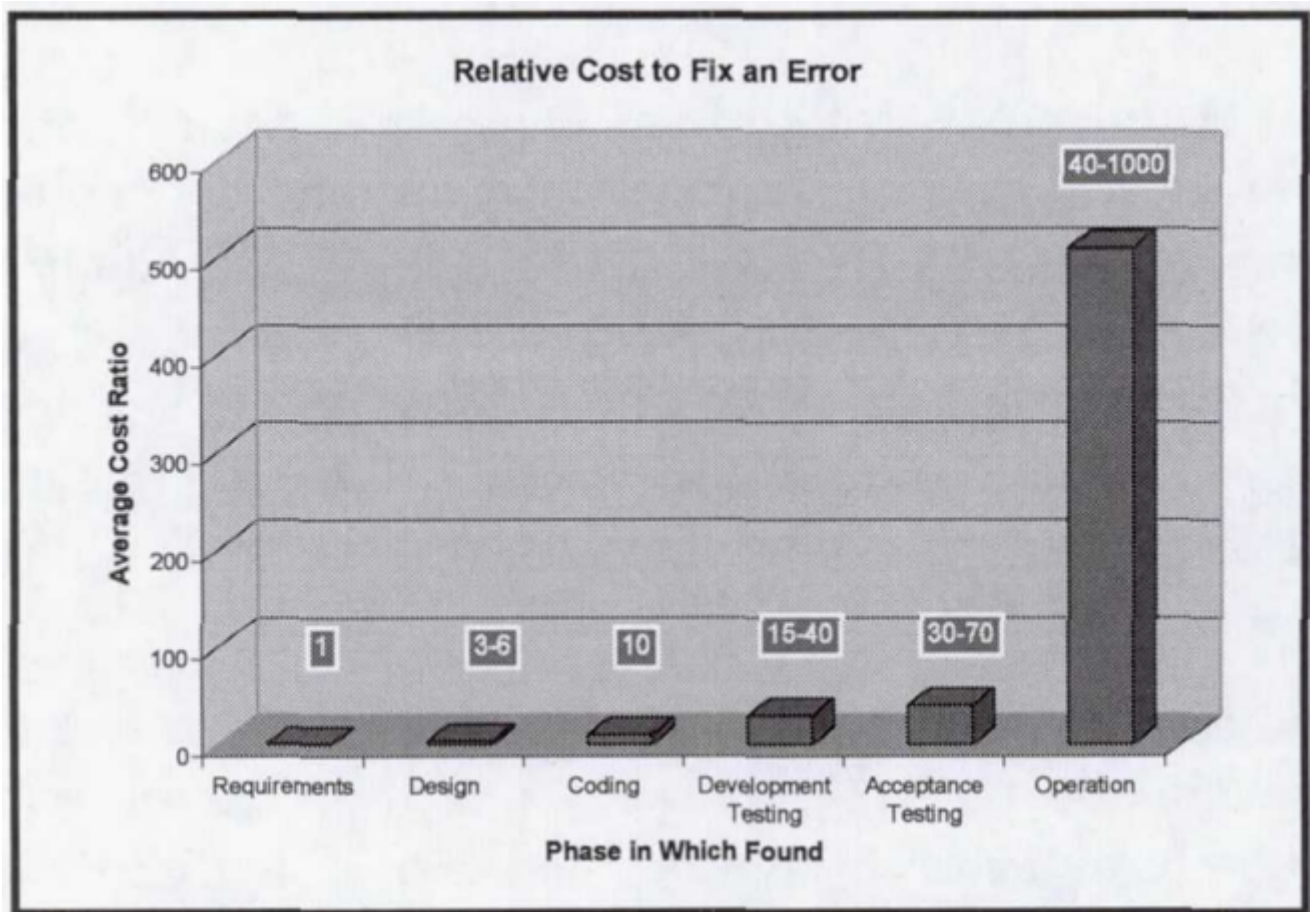


Figure 5.1: Relative Cost to Fix Software Errors per Life Cycle Phase [86]

Figure 5.1 demonstrates such a phenomena in the cost of software error correction per life cycle phase [86]. As a result of this, there is a great deal of cost to be avoided (or savings to be made) by investing resources into the system architecting process early to ensure errors are avoided. Among others, such resources can include; project schedule allocated for developing the system architecture, expertise in personnel, and development and application of robust engineering practices that are more likely to give a high quality output REF?. These robust engineering practices are part of the focus for the application of MBSE.

As stated in ISO 15288, Systems and software engineering — System life cycle processes standard [81], the outcomes of the system architecting process are:

- A. The problem space is refined with respect to key stakeholder concerns, context, and

perspectives

- B. Alignment of the architecture with applicable policies, directives, objectives, and constraints is achieved
- C. Concepts, properties, characteristics, behaviours, functions, or constraints that are significant to architecture decisions of the system are allocated to architectural entities
- D. Identified stakeholder concerns are addressed by the system architecture
- E. Traceability of system architecture elements to key architecturally-relevant stakeholder and system requirements is established
- F. Architecture views and models of the system are developed
- G. System elements including their interfaces with each other are defined
- H. Enabling systems or services needed for system architecture definition are available

The outcomes defined by ISO 15288 are addressed in a number of ways across existing methods of system architecting. The majority of typical MBSE methods achieve then by offering formalization of system architecture concepts. As seen in chapter 4, MBSE methods set out a series of steps to be completed in-order to arrive at a complete system architecture. Alongside the selected MBSE language and ontology, that define the types of element included in a system architecture, as well as their inter-relations, this provides the necessary concepts to complete a system architecture model. see Table 5.1 for a comparison between two MBSE methods approaches' to achieving the system architecting process outcomes.

Formalization of the system architecting process facilitates better communication in two ways [87],[88]. firstly between stakeholders such as the system architects on the project and project management, as there can be shared expectations wrt. documentation, nomenclature and overall meaning. Secondly for digitization, allowing computer programmes to ingest the system architecture information, allowing the automation of certain tasks.

Table 5.1: ISO 15288 System Architecting mapping to example MBSE methods, [89], [50]

	Outcome	Arcadia Step		SEAM Module	
A	The problem space is refined with respect to key stakeholder concerns, context, and perspectives	Perform	Cus- tomer Oper- ational Need Analysis	Requirements	
B	Alignment of the architecture with applicable policies, directives, objectives, and constraints is achieved	Design	Physical Architecture	Logical/Functional Architecture	Archic- itecture
C	Concepts, properties, characteristics, behaviours, functions, or constraints that are significant to architecture decisions of the system are allocated to architectural entities	Design	Logical Architecture	Operations, Logical/Func- tional Architecture	
D	Identified stakeholder concerns are addressed by the system architecture	Perform	System Need Analysis	Logical/Functional Architecture, Requirements	Archic- itecture
E	Traceability of system architecture elements to key architecturally-relevant stakeholder and system requirements is established	Design	Logical Architecture	Logical/Functional Architecture, Requirements	Archic- itecture
F	Architecture views and models of the system are developed	Design	Logical Architecture, Design Physical Architecture	Logical/Functional Architecture	Archic- itecture
G	System elements including their interfaces with each other are defined	Design	Logical Architecture, Design Physical Architecture	Logical/Functional Architecture	Archic- itecture
H	Enabling systems or services needed for system architecture definition are available	Perform	Cus- tomer Oper- ational Need Analysis	Mission	

5.1.2 System Architecting Decision Support Frameworks

Beyond the typical MBSE methods and architecture frameworks, a number of specialized system architecting decision support frameworks have been developed [90], [91], [92], [93], [94], [95]. While 'primary' MBSE methods focus on the concepts necessary for the complete description of a single system architecture, these frameworks provide further concepts in their formalization of the variability of alternative system architectures. This generally involves a concept of design decision, that contains the information related to a specific choice made during the architecting process, a concept of constraints, that limits the allowable design decision outcome and finally a concept of design selection, that provides a means of choosing between the possible outcomes for a decision. Table 5.3 and Table 5.5 give a brief description of the various concepts available in two such decision support frameworks, loosely mapped to the three concepts described above [89], [50].

When considering the applicable domains of formalized system architecting frameworks, there are typically two types; domain-agnostic and domain-fixed. Domain-agnostic frameworks leverage no specific domain knowledge and instead simply provide the architect with the abstract concepts necessary for arriving at a system architecture, allowing them to apply their own domain knowledge on top of the abstract formalization [90], [92], [94], [95]. This has the benefit of being applicable to virtually any domain that is complex enough it is necessary to apply the system architecting process (and systems engineering in general). However these frameworks rely entirely on the architect's understanding of the domain in which they are working. The system architectures developed using these frameworks must be populated almost entirely by hand, as the frameworks used will make little to no assumptions about the details of the system, therefore no shortcuts can be made in this regard and opportunities for automation are limited.

Conversely, domain-fixed architecture frameworks do make detailed assumptions about the nature of their respective domain and the systems that operate in them, such as [93] and [91]. [68] could also be considered under this category, however it deals more with the generation of design concepts, rather than formal system architectures. This approach facilitates higher levels of automation in many of the tasks related to the architecting process, even upto component selection as with [68]. However this comes at the cost of greatly reduced scope of applicability. Further to this, highly automated methods, such as [68] run the risk of limiting the visibility of the decision making process, by converting much of it into a black box input/output procedure. This can make justifying the chosen

Table 5.3: Types available in the Architecture Design Space Graph (ADSG) and their related variability concepts, see [96] for further detail

Type	Fundamental type	Source	Related Variability Concept
Function		Design space definition	Design description
Concept	Function mapper	Design space definition	Design description
Function decomposition	Function mapper	Design space definition	Design description
Component	Function mapper	Design space definition	Design description
Component instance		Component definition	Design description
Attribute	Source	Component definition	Design description
Attribute value	Target	Component definition	Design description
Output port	Source	Component definition	Design description
Input port	Target	Component definition	Design description
Option-decision		Rule-based insertion	Design decision
Permutation-decision		Rule-based insertion	Design decision
No-operation		Component definition	Design decision
Grouping		Component definition	Design decision
Boundary incoming		Always present	Constraints
Performance metric		Design space definition	Means of design selection
Design variable		Design space definition	Means of design selection

Table 5.5: Types available in the Model Based Product Line Engineering (MBPLE) framework and their related variability concepts, see [96] for further detail

Group	Type	Label
Product Line Features	Feature	Design description
	Mandatory Feature	Design decision, Constraints
	Optional Feature	Design decision
	Alternative Feature	Design decision, Constraints
	OR Feature	Design decision, Constraints
	Feature Configuration	Means of design selection
Product Line Shared Assets	Shared Assets	Design description
	Common Asset	Design description

architecture to other stakeholders challenging, as there may be less trust in a black-box process such as this, compared to a transparent more traditional one [97].

5.1.3 System Analysis and MDAO

Several architecting framework examples in literature include links to system analysis tools that facilitate assessment of the architecture's performance and compliance wrt. requirements. MORE TO WRITE here

5.1.4 The Knowledge Gap

With the various approaches discussed above it becomes clear that there is a knowledge gap wrt. system architecting methods that offer both domain flexibility and capacity for guidance and automation. While these capabilities have been achieved separately, the following work has developed a method of system architecting, and associated tooling, that draws on domain knowledge to intelligently guide the architect towards high quality architectures, while maintaining clear traceability to requirements and insight into the reasoning performed by the associated tool. The domain knowledge elicitation process is also considered as part of this method and enables flexibility across domains. NOW LIST CHAPTER SUB SECTIONS

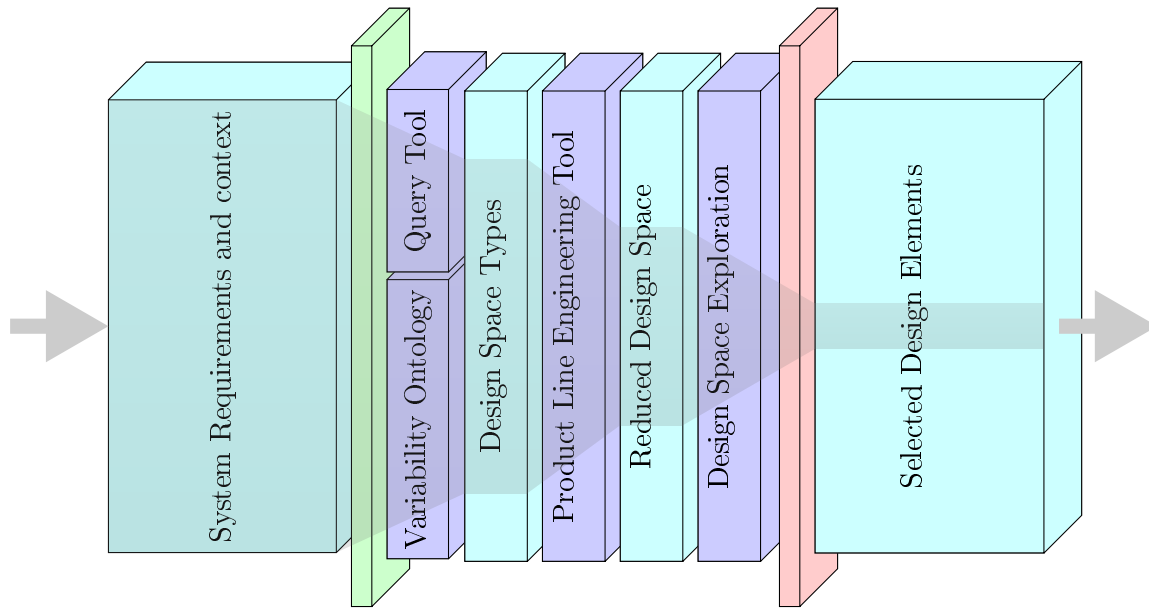


Figure 5.2: Overview of the framework elements and its context

5.2 Method Definition

To facilitate flexible guided system architecting and number of techniques and tools have been developed and are presented in the following sections

5.2.1 Process and Overview

Figure 5.1 shows the tools and top level process of system architecting when using a domain knowledge base in such a guided manner. Starting with the requirements, stored in a requirements model, the architect then can begin modelling the architecture to what ever degree they wish or immediately start querying the knowledge base. When querying, the system architect is provided with sets of recommendations of what to include in the architecture. The recommendations were intended to cover a wide range of possibilities, such that the system architect can downselect these via some means for design space exploration, i.e. cast the net wide rather than deep. The architect can then follow this design loop of querying the knowledge base with the current architecture and requirement models, to formulate design problems and executing each design space exploration to select modifications and update the architecture model.

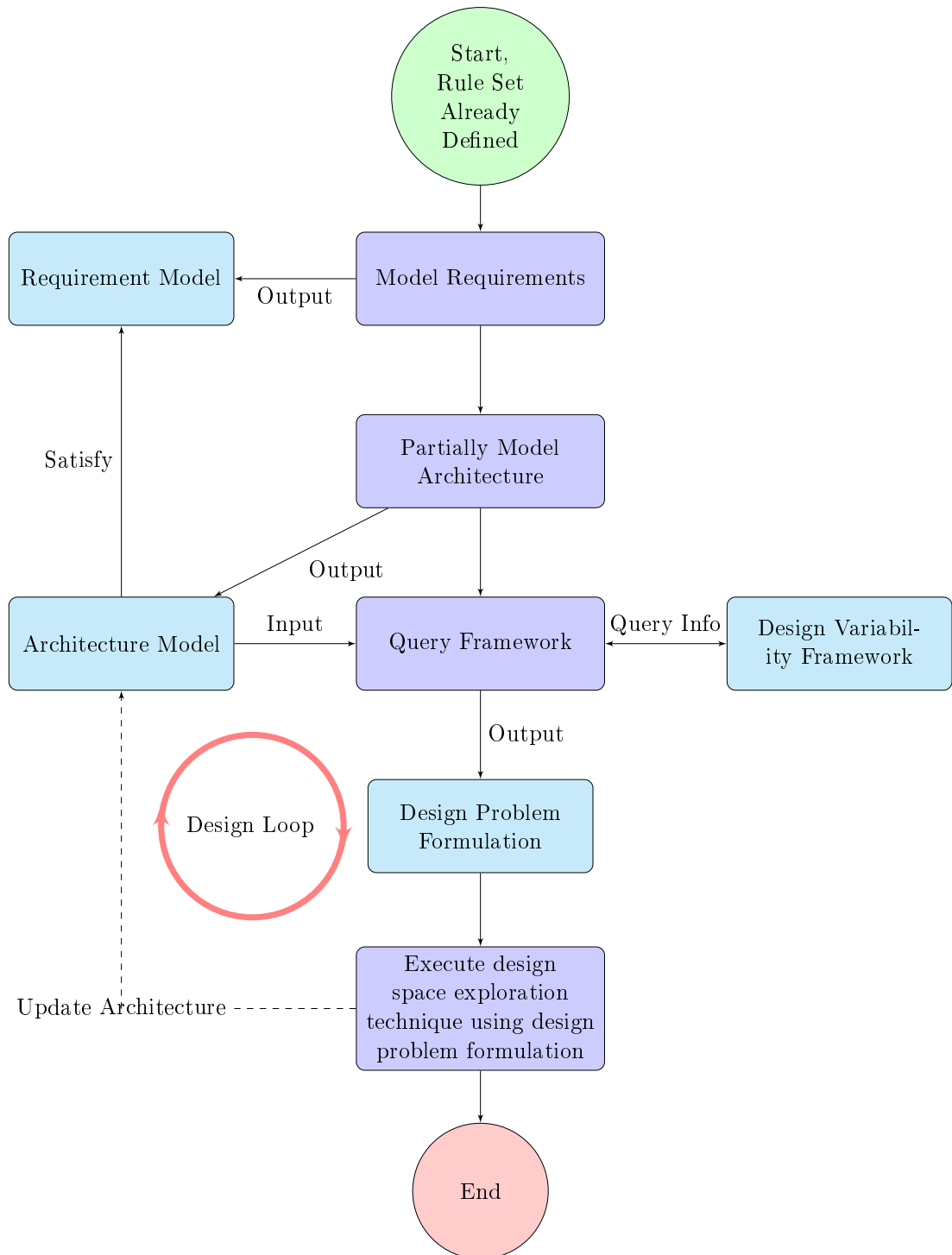


Figure 5.1: System architecting process, making use of a domain knowledge base

5.2.2 Assumptions and Static Ontology

This approach to system architecting relied on a number of assumptions about the nature of a system architecture and its context. These were derived from various sources, the first of which was [96]. The assumptions were as follows VERBATUM:

- A.1 From [96]. Stakeholders and their needs have been identified, and from that, the requirements have been defined: it will be assumed that the requirements definition process happens before the architecting process [98]: by the time the system architecting process is started, a well-defined base exists for starting to define the system's main functions. It is recognized that there can be iterative interactions between the requirement definition and architecting phases, but there will at least be some starting point to base the architecture design space on.
- A.2 From [96]. Design choices in system architecting are sufficiently represented by decisions related to function fulfillment, component characterization, and component connections: literature shows that these three categories of decisions cover all of the general design choices of system architecture design problems [98] [99].
- A.3 From [96]. It is possible to quantitatively evaluate the performance of architectures: this allows the use of systematic design space exploration techniques like a Design of Experiments or optimization for finding the best architecture to solve the problem at hand.
- A.4 A system architecture can be described as a set of connected elements
- A.5 Each of the architecture elements can be assigned one of the following fundamental types; Requirement, Function, Component, Mode, Parameter.
 - A. From [96]. Each function is fulfilled (performed) by one component
 - A. From [96]. CHECK Components can derive (induce) additional functions
 - A. From [96]. Connecting components is done after mapping components to functions: a need for a connection between components can therefore not be used to include these components in the architecture, only the need for fulfilling functions can.

- A. Each of the elements can be assigned further types or labels, related to the application domain and characterized by their set of potential relationship-neighbour type triples.
- A. A non cyclic hierarchy of types can be formed, where one type may be a subtype of another, while labels do not follow a hierarchy
- A. Requirements and functions can be labelled, whereas components, modes and parameters can be typed.
- A. All architecture elements have traceability, directly or indirectly, back to requirements
- A. The set of possible architecture element types in a domain depends (directly or indirectly) on the labels on requirements given as input to the architecting process
- A. Model Hierarchy has been ignored
- A. Each requirement should have a unique set of labels, unless referring to different levels of performance or conditions. Not enforced for training data (slightly reduces performance). This puts a floor on the level of detail used for the out of training domain testing models
- A. Functions inherit labels from the requirements they satisfy

Once these assumptions had been established, a static ontology was formed to further refine the fundamental types described above. While this approach focussed on deriving domain specific details of the system architecting process, this static ontology was expected to remain the same regardless of the application domain. This ontology is shown in Figure 5.2.

This ontology was constructed in reference to existing MBSE methods, with the aim of introducing the minimum number of concepts while maintaining compatibility with typical MBSE practice and ontologies. As a result the 5 fundamental types in this static ontology can be mapped to existing concepts in typical MBSE ontologies. For example, the Arcadia method and language follows an ontology(or meta model) that includes the above types alongside various other concepts [100]. SysML concepts are quite abstract compared to the concepts in the static ontology used here. But generally speaking, SysML blocks can take the role of components, SysML states take the role of modes

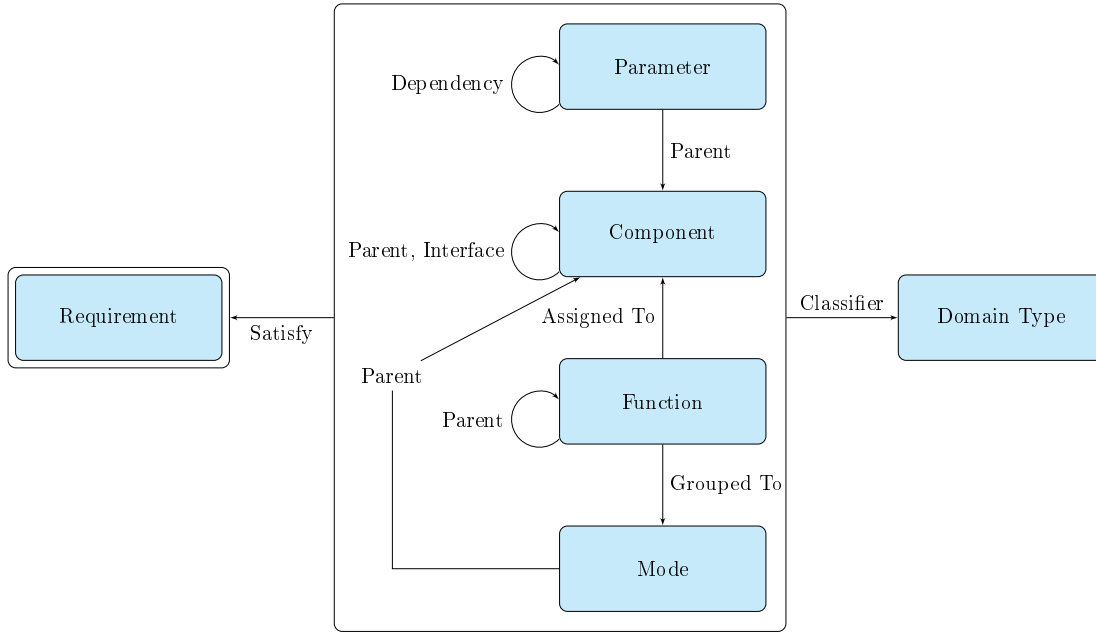


Figure 5.2: Static ontology for system architecting

and activities take the role of functions. Besides this, requirements already exist in SysML. MBSE methods that utilise SysML commonly then build on the core SysML meta model, via stereotypes, to include equivalents of components, functions and modes. For example the OOSEM logical stereotype (on top of the SysML base class) can be mapped to the components type [38] and SEAM includes specific concepts of functions, modes and components (as Logical Elements) [50].

The ontology allowed the modeller to describe a hierarchy of components alongside a hierarchy of functions, essentially the logical and functional architectures seen in MBSE methods like ARCADIA [100] or SEAM [50]. At each level of the system, the functions were assigned, or 'clustered' together to form the components and grouped to form modes. The process of clustering functions into elements is described in literature [101] as drawing together functions with related inputs and outputs, thus demonstrating closely related functions and providing a candidate logical architecture. For example all functions related to capturing an image would be 'clustered' into a single camera logical element. A mode is defined as 'a collection of available functions that allows the system to perform specific tasks' [102]. This means that the modes of a particular logical element can be developed by 'grouping' functions into groups. For example all functions that need to be performed while imaging would be grouped into an imaging mode.

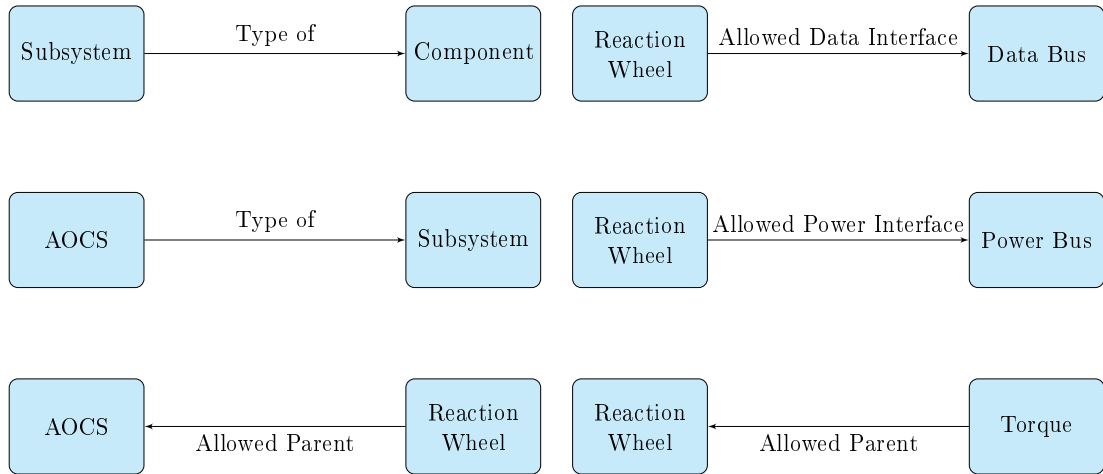


Figure 5.3: Examples of domain specific design rules

Besides this, the ontology provided the concept of parameters that can be owned by a component, to include any necessary information for performing analysis or demonstrating compliance with performance constraints set out by a requirement. It was possible that functions or modes would benefit from being further defined by adding parameters, but this was not been included in the ontology for simplicity. Instead, the component that was associated to that function or mode (by a assigned to or parent relationship) would be modelled as the owner of that parameter. The ontology allowed any component, function, mode or parameter to satisfy a requirement, ensuring high flexibility in this regard.

5.2.3 Domain Knowledge Base and Dynamic Ontology Layer

With the static ontology above defined it was then necessary to define a second, dynamic, layer of the ontology that would be specific for each application domain, thus forming the knowledge base for that domain. Following assumption ADD REF, the fundamental types shown in Figure 5.2 could be refined into further types, each with their own allowed relationships with other types in the domain. This information would form new design rules in the knowledge base. Figure 5.3 gives a few examples of these rules for the domain of a Spacecraft. The process of identifying the rules in the dynamic layer of the knowledge base, for a particular domain, is covered in a later section. NOTE danger of picking technology here?

5.2.4 Knowledge Base Queries

The process of querying the knowledge base was the step that provided the system architect with the design problem formulation necessary to select a solution that matched the design rules included in the knowledge base.

Building on the static ontology in Figure 5.2, a number of query types were defined, that also referenced existing MBSE methods. The query types were intended to be roughly applicable to particular steps in other MBSE methods and their details are shown in Table 5.1 and sections Section 5.2.4,Section 5.2.4,Section 5.2.4 Section 5.2.4,Section 5.2.4.

Table 5.1: Query Types and details

Query Type	Requirements	Design Variable	Dependant Variable	Suggested Design Variable	Suggested Dependant Variable	Additional Model Elements
Requirements to Functions System Analysis	✓	✓	×	×	×	✓
Functions to Components System Analysis	✓	✓	×	×	×	✓
Functions to Modes System Analysis	✓	✓	×	×	×	✓
Functional Decomposition System Analysis	✓	✓	×	×	×	✓
Internal Interfaces Analysis	×	✓	×	×	✓	✓
Parametric Analysis	✓	✓	✓	✓	✓	✓

Requirements to Functions System Analysis

Requirements: - when considering requirements: (initial discovery) (4) Design variable: - when considering design variables: (initial discovery) (5) Dependant variable: - when considering dependant variables: (initial discovery) (6) suggest dependant variable: - when considering new dependant variables from model (7) - as related to design variables via interface relationships, not already interfaced to in design: (8,s) - as related to design variables via hard dependency relationships (9,s) - as related to requirements via satisfy relationships (10,s) - as related to requirements via satisfy relationships in knowledge base (11) suggest design variable: - when considering new design variables from model (12) - when considering new design variables from suggested dependant variables (13) - when considering new design variables from currently selected dependant variables (14,s) additional model elements - when considering additional model elements: (18) - as possible children of current design elements (19) - When considering requirements (1) - When considering design variables (2,s) - When considering dependant variables (3,s) - as possible dependencies of current design elements (20) - When considering requirements (1) - When considering design variables (2,s) - When considering dependant variables (3,s) - as possible elements satisfying requirements (21) - When considering requirements (1) - When considering design variables (2,s) - When considering dependant variables (3,s)

Functions to Components System Analysis

Functions to Modes System Analysis

Functional Decomposition System Analysis

Internal Interfaces Analysis

Parametric Analysis

5.2.5 Design Space Exploration

The design exploration process was considered a distinct step in this process and should, by following the design problem formulation output by the knowledge base queries as shown in Figure 5.1, select a specific design in the design space. This means all the possible abstract elements types included in the design problem formulation output by

the knowledge base query have either been selected or discounted and all parameters relevant to the design problem have had a value selected.

CHAPTER 6

DEMONSTRATING VARIABILITY FRAMEWORK GUIDED SYSTEM PARAMETER OPTIMIZATION OF EARTH OBSERVATION SPACECRAFT

Model Based Systems Engineering (MBSE) is an interesting alternative to traditional systems engineering methods. Instead of using electronic documents to record system information, MBSE uses a unified and coherent system model. Trade-offs are a major element of a space systems engineer's role in early system design. This can be a particularly challenging process in the domain of spacecraft, as the system designs are often very complex and the constraints can be difficult to characterize. There has been little previous research on the use of MBSE as a design space exploration tool or in support of trade-offs. This paper investigates the potential to use MBSE for design exploration and to understand trade-offs, through the creation of a new toolset including a SysML profile. The tool draws on generative design (allowing automatic guided generation of a multitude of design alternatives) and system optimization to rapidly generate and assess new designs using interactive analysis and visualizations. Techniques such as surrogate modelling, genetic algorithms and robustness measurements will be available in the toolset. The toolset was applied to a design scenario aiming to improve the trade off and design

selection process for LEO Earth observation satellites. The upcoming ESA TRUTHS space mission was used as a case study and the design process was recorded and compared to a manual design exploration approach. The toolset was found to reduce the design exploration time by 38% to 96% , allow exploration of more designs in an equivalent time and provide better quantification of the relationships present in the design space, all without drops in selected design quality. For now, the toolset can only perform parameter variation in the design exploration and future work is expected to extend this to higher levels of variability. The study also discusses how the MBSE toolset could be applied to other missions, offering the same advantages to all early phase spacecraft designers.

6.1 Introduction

MBSE aims to provide a “single source of truth” for a project [1] that forms the communication platform for all stake holders. By doing so, it avoids inconsistencies in system information commonly experienced with traditional systems engineering, where any change to the design requires updates in several documents in disparate locations. Completeness and communicability [2] are two further benefits reported by literature.

While MBSE has been shown to successfully describe a system baseline, its applicability could be extended by introducing tools for rapid exploration of the design space. Currently, designing in an MBSE environment is a heavily manual and time consuming process, relying on engineers to use their expertise to create a single or small set of candidate designs. A large portion of the modelling effort is directed toward verification of the limited candidate design set, with only a small capacity left for iterating on these or exploring a wider range of options. This means there is little confidence that the selected design is the ‘best’ rather than ‘satisfactory’ and these challenges are compounded in complex and nuanced scenarios with many variables.

The primary language used by MBSE, SysML, has limited native support for describing how elements of a design may change from one design option to another [3], [4], and many common MBSE methodologies, such as the OOSEM [5] do not address trade-offs or iterating of the design in detail. MBSE has been lacking in this area as until now most projects have relied on manual iteration and exploration of designs.

In previous work, extensions of the OOSEM trade-off approach that made use of new SysML stereotypes to create decision points have been proposed [3]. From these decision

points Constraint Satisfaction Multi-objective Optimization Problems (CSMOP) can be generated and solved. CSMOPs are an example of optimization problem that includes constraints and multiple objectives which may contradict one another. To perform the optimization itself, solvers such as the python PyOpt were used. This was an example of annotative modelling, whereby the baseline and variability of a model is described in a single over specified model [6]. Further methods of performing trade-offs and design space exploration exist, but their application as part of an MBSE methodology are sometimes limited. The following are examples and are discussed in the next section: multi domain system optimization, surrogate modelling and generative design.

Multi-domain system optimization is an iterative or explorative process whereby an objective function is minimized or maximized as part of a multi-domain system model. When carrying out complex multi-domain trade-offs, it is becoming increasingly common to formulate the decision as a mathematical problem. This allows the use of optimization algorithms such as examples of meta-heuristics like genetic algorithms to come to a decision, generally in less time and for less cost [7]. Several examples of functions used for benchmarking such optimisation algorithms exist. This study used the Rastrigin function [8] for this task, as discussed later.

Surrogate modelling has been proposed for design space characterization when it is expensive to evaluate the objective function at every iteration of the optimization [9]. This can be common in spacecraft design. The method involves making an approximated, cheap to evaluate, “surrogate model” of the original that can interrogate its own design space and thus gain an understanding of the original model’s design space.

Generative design (GenDes) provides the designer with many design alternatives created via an automated, algorithmic process, generally starting from mission requirements and other design constraints [10]. While most commonly applied to shape/form optimization [11] [12], some application of GenDes to spacecraft has been investigated, largely focussing on manufacturing and detailed component design [13]. However it has had very limited application with MBSE for system architecting.

If MBSE tools could be extended to include variability modelling, this would present an ideal opportunity to apply the methods discussed above to allow the use of MBSE as a design exploration tool.

The aim of this study was therefore to assess a new toolset and semantics for SysML, based on a new profile (SysML’s inbuilt extension mechanism) for the purpose of design

space exploration. This would support design space exploration and with the aim of achieving the following requirements:

- The design space exploration process shall map as much of the design space as possible
- The process will identify optimal designs quickly, compared to manual design space exploration
- Any additional model elements, such as diagrams or blocks, should be as reusable as possible
- The results of the exploration should be provided to the designer in a visual and/or interactive fashion

The rest of this paper is split as follows; section 2 describes the method followed during the study, the setup of the toolset, how the TRUTHS mission was modelled and its design space explored, section 3 presents results, section 4 discussion, and finally section 5 covers conclusions and future work.

6.2 Method

The MBSE tool chosen was Dassault Systems' Cameo Systems Modeller © [14], this being one of the most widely used MBSE tools in industry. The Cameo Simulation Toolbox was used as the basis for running the design space explorations. Fig.1 shows the expected steps taken when using the toolset.

6.2.1 Design Space Stereotypes

Similarly to [6], an annotative approach to variability modelling was taken (with more emphasis on continuous parameter variability). Therefore in order to capture information required to identify variability in a model, profile diagrams were used to define new SysML stereotypes. SysML stereotypes are used to extend existing SysML elements with new constraints and properties. These new stereotypes are shown in Fig. 2.

The design variable declaration stereotype can be applied to a class (or block) in the model to identify how an element of the system should be actively varied during the design space exploration. Examples of this may include orbit parameters, solar array areas and

other elements of the design that can be directly controlled by a designer. In this way the design variables were the independent variables of the trade space exploration.

The dependent variable declaration was used to identify important elements of the system that should be recorded at each design point in the exploration. This was useful in linking mission requirements relating to these elements that must be met for a valid design. In this way it was possible to apply constraints to the design space, directly related to mission requirements.

The objective declaration was used to identify aspects of the system that have some goal related to them, i.e. to be maximized or minimized or play a role in a trade-off. When multiple objectives are defined, a weighted sum of all the objectives is taken to condense the objectives down to a single objective function. This stereotype could be used in a similar way to the existing stereotype “Measure Of Effectiveness” (MOE). The main difference being that the new objective stereotype also contains a weight for its objective, which would have to be defined elsewhere should the MOE approach be used.
ADD FIGURE TWO HERE

6.2.2 Toolset Workflows

As previously discussed, the Cameo Simulation Toolkit was used to execute design space explorations. This allowed a SysML based approach that had two advantages over using a separate language, such as the Orthogonal Variability Modeling (OVM) language [15]. First, it meant that design explorations could be performed without setting up a completely separate tool with a new language for variability. As a result any existing system simulations already setup with Cameo Simulation Toolkit could be directly integrated with the design space exploration process. For example, system operations simulations could be dropped into the design space exploration simulation and executed for each design point, without the need for any external software interfacing. Second, this approach allowed the use of SysML as the language to describe the design exploration itself. SysML Activity diagrams were used as a basis to implement a custom SysML diagram called a “Workflow” that could be executed by Cameo Simulation Toolkit and thus drive the design space exploration. Opaque actions were added to each workflow diagram containing the code necessary for the exploration, most of which is written in either JavaScript or MATLAB. All code used in the workflows was developed to be as generic as possible and could work with any model, assuming the custom stereotypes

had been correctly applied. This allowed the design exploration to be captured by the model itself, allowing better traceability of the decision-making process for the choice of a particular design point. Fig. 2 shows an example workflow diagram used to drive the design exploration.

Four main workflow diagrams were created. First, a design space exploration workflow (Fig.3) that uses a basic generative design approach to drive the model through a set of designs, using uniform sampling across each design variable. All the necessary analysis could be run within the Design Point Analysis Sequence Activity. Second, a surrogate model creation workflow that uses collected design points to build mathematical approximations of the model using Radial Basis Functions (RBF) [16]. RBF's were used as they are very versatile both in terms of the relationships they can approximate and the number of dimensions they are applied to. This makes them applicable to more situations for example than polynomial approximations that are limited by their order. An optimization workflow was created that uses a MATLAB genetic algorithm to optimize the surrogate model and suggest an optimal design point. Finally, a design space visualization workflow was implemented making use of MATLAB's inbuilt graph generation tools. 3D plots were used to display the design space, with plots for the behavior of dependent variables and the objective function. The designer could select up to two design variables to plot against for each one. Figures 16,18,19,20,21 and 23 were generated using this workflow.

These four workflows could be applied separately or in a complementary fashion. The generative design workflow could be used simply to generate data about different designs or have the designs fed straight into the surrogate model creation workflow to identify underlying relationships. The designer may then choose to input the surrogate model into the optimization workflow to receive a suggested optimal design point. Finally the design space visualization workflow could be used at any stage of the process to display collected design space data, meaning the designs generated by the design exploration workflow, the surrogate model approximations and the suggested design provided by the optimization workflow.

Once the toolset SysML customizations and workflows were complete, the next step was to apply them to a system model and assess its value in exploring the design space.

FIGURE 3 GOES HERE

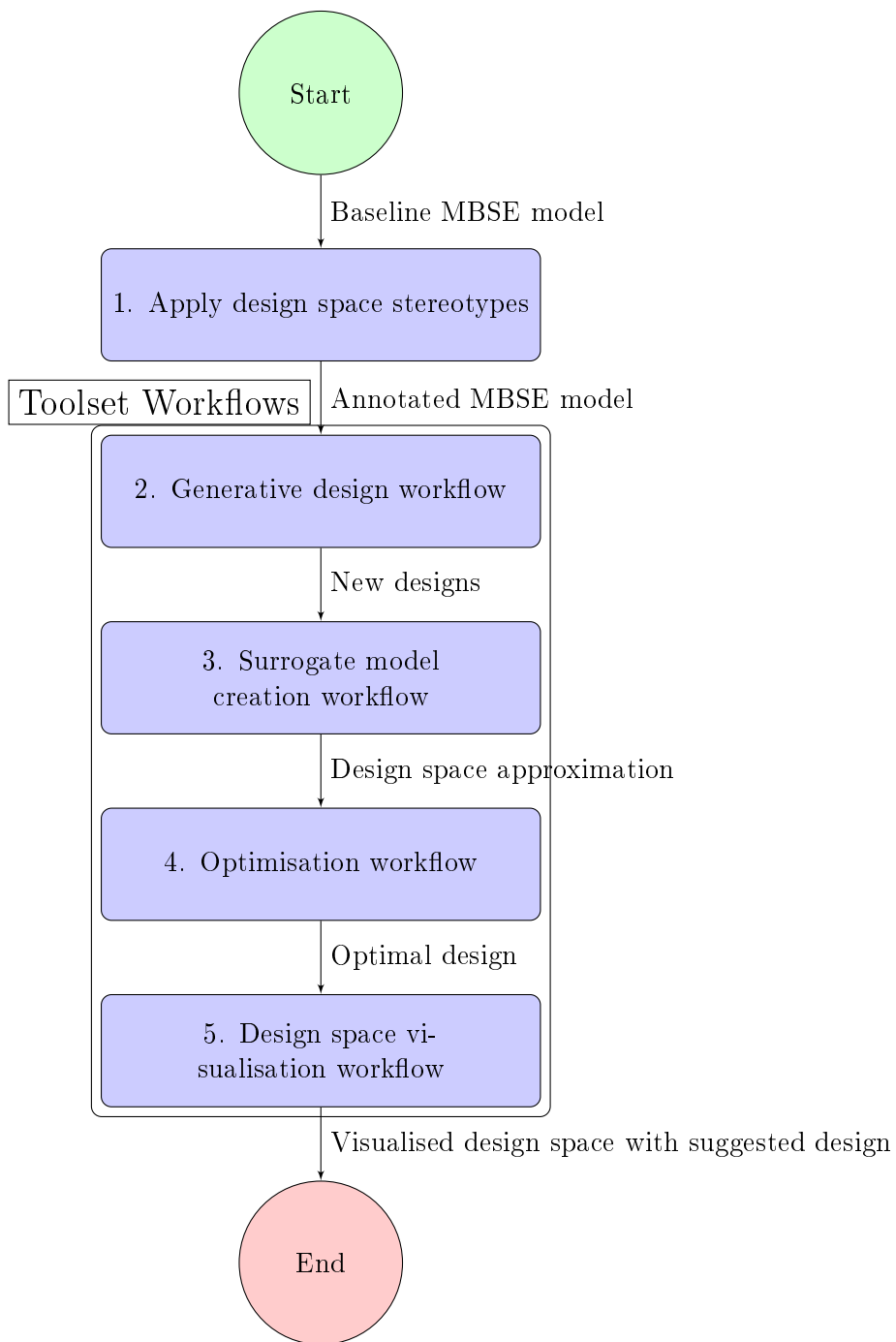


Figure 6.1: Steps taken when using the toolset

6.2.3 TRUTHS Test case

This section will present the TRUTHS mission and how it was used to assess the new toolset. The Traceable Radiometer Underpinning Terrestrial and Helio Studies (TRUTHS) is an upcoming ESA mission due to launch in 2029. TRUTHS mission aim is to collect a wide range of measurements from a number of sources in order to provide calibration data for Earth observation and climate studies [17]. A basic breakdown of the system elements focused on for this study may be seen in Fig.4. The Buffer elements belong to model elements that have data storage for payload data. The Hyperspectral Imaging System (HIS) was the primary payload, collecting data from the Earth's surface when sunlit. The OnBoard Calibration System (OBCS) collected data from the Sun and Moon (alongside the HIS on occasion). The two payloads field of view were 90° apart. The AOCS subsystem handled each of the maneuvers that TRUTHS will perform, which are complicated by the different pointing directions of the two payloads.

FIGURE 4 GOES HERE

Baseline Model

To allow an assessment of the toolset in a controlled application, a model of TRUTHS that could be run with various forms of analysis first needed to be completed. This model followed the spacecraft early analysis model framework presented in [18]. The model used the breakdown structure in Fig.4 as the logical architecture and a functional architecture following the system mode diagram (SysML state machine diagram) shown in Fig.5. The functions included in this functional architecture model were all related to imaging and the various imaging maneuvers TRUTHS will perform, as discussed later.

FIGURE 5 GOES HERE

It was decided to build two analysis tools, to demonstrate the toolset's behavior across different analysis approaches. The first analysis tool was a detailed time-based simulation of the system's operations, that would provide an opportunity to demonstrate the toolset's capability to identify complex and subtle relationships that would not otherwise be obvious to the designer. As Cameo's simulation toolkit allowed direct execution of SysML, the logical/functional/operational model discussed above could be executed and propagated through a time period during the mission. The second analysis tool was a MATLAB script that executed a simplified model of TRUTHS operations, taking its

inputs from the SysML model. This analysis was quicker to run, therefore making it more useful for performing more complex design space explorations, compared to the slower, more detailed analysis tool. In practice, this toolset proposes using the simplified analysis tool to identify the best candidates, which are refined and verified through the detailed analysis tool.

Data handling related operations were chosen as the focus of these simulations, including target accesses, payload data acquisition and downlink. Maneuvers required for imaging were also considered, to gain a representative picture of target imaging start and stop times. Fig.6 shows a simplified execution cycle of these operations through each orbit. As a result only the functions related to these operations were included in the functional architecture model.

FIGURE SIX GOES HERE

To help identify the timeline of operations during the simulations (with both analysis tools), a custom interface to Systems Tool Kit (STK) orbital modelling software from the Analytical Graphics, Inc. (AGI) [19] was implemented. This ingested the satellite, instruments, targets and ground station information in the SysML model into an STK scenario and calculated the access times between instruments and targets and the satellite and ground stations. The interface also inserted maneuvers to the timeline at the appropriate times during in the simulated operations, however this was only done when using the detailed analysis tool.

The simulations were validated by comparing both the detailed and simplified simulation's output results to results generated using Airbus' own in-house tools as shown in figures 12 and 16. The key results were maximum data latency and maximum onboard data, as these are used to drive the onboard storage sizing and downlink capabilities.

6.2.4 Controlled Design Space Explorations

Three design space explorations were performed on the baseline TRUTHS model (except for the third) using the toolset, these were recorded, then the equivalent explorations were undertaken manually. This manual process was performed by the same modeler who had setup the baseline model, therefore were able to draw on their own understanding of the design (in much the same way as would be done in industry). This allowed a near direct comparison between the two processes and an objective assessment of the benefits and limitations of the developed toolset. Fig.7 shows these basic steps taken for each design

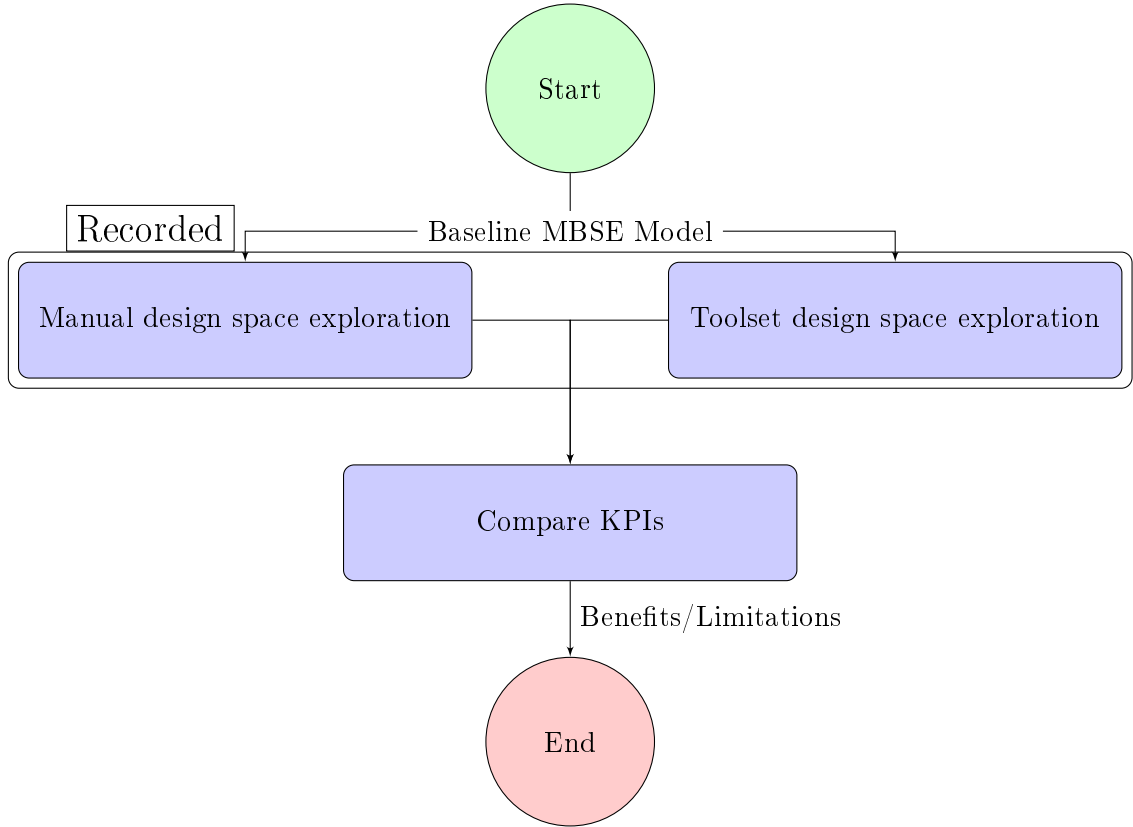


Figure 6.2: Steps undertaken for each design space exploration

space exploration experiment.

The comparison was achieved by selecting a group of Key Performance Indicators (KPI's) by which both the generative (toolset) and manual processes were measured. These KPIs were concerned with assessing the scope, time and cost of the two processes. DiscussionS with Airbus and ISO22400 [20] were used to inform the selection of KPIs. These KPIs were defined as follows:

Number of Model Edits/Additions/Removals (Cost)

As each change to the model is done manually, this KPI forms a metric for the amount of human effort put into each process and therefore is a proxy for cost. Besides this, model edits are a common source of mistakes that can rapidly drive up cost.

Total Time For Exploration (Time/Cost)

Equivalent to actual order execution time in ISO22400. In almost any case, the time taken to perform an action will have some impact on its cost, either in terms of a paid engineers' time being spent or processing time on a computer.

Setup Time (Time)

Equivalent to 'actual unit setup time' in ISO22400, and as the setup was the only example of manual work in both processes, this is also similar to actual personnel work time. During discussion with Airbus, the setup time of analysis and tools was identified as a major source of delays. This can be particularly troublesome as, unlike running the analysis itself, it relies heavily on manual effort rather than computer time. The setup time here does not include the time taken to setup the baseline model, as this was the same for the two processes, only the setup time of the design space exploration.

Number of Designs (Scope)

Somewhat equivalent to 'inspected part' in ISO22400. The number of designs generated will provide an estimate of the coverage of the design space as a whole. While the true coverage of features and relationships in the design space will remain unknown, this is still a useful metric to compare the two processes against each other.

Performance of Chosen Design (Scope)

The output of each process, the final selected design, was assessed using the objective function defined for each design space exploration.

Difference of Selected Design to Average Performance for Explored Design Space

The difference between the final chosen design and the average performance (objective function value) of all collected design points across both processes. This gave an indication of how much better or worse each process was at selecting designs compared to simply picking a single design point with no exploration of the design space.

TABLE 1 GOES HERE

The recording of each process required to calculate these KPIs was performed via logging

that identified the time, subject, duration and resource (human/computer) for each of the events shown in Table 1.

The execution sequence of toolset workflows during the design space explorations was as follows. First, the generative design workflow (Fig.3) was run to generate a set of design points. Second, the surrogate model creation was run and finally, the optimization workflow was run to arrive at a suggested design point. When performing the manual equivalent of these explorations, the model was updated by hand to a new design point and the same analysis sequence was run, results collected in an excel spreadsheet and the cycle repeated for each design point.

Design Space Exploration Setup

Three controlled design space explorations were undertaken and are detailed in Tables 2 to 4

SPACE ARCHITECTURE GENERATION USING LLMs

CHAPTER



DISCUSSION

CONCLUSION AND FUTURE DIRECTIONS

A P P E N D I X



APPENDIX

Bibliography

- [1] P. Leserf, P. de Saqui-Sannes, and J. Hugues, “Trade-off analysis for SysML models using decision points and CSPs”, *Software and Systems Modeling*, vol. 18, no. 6, pp. 3265–3281, 2019, ISSN: 16191374. DOI: 10.1007/s10270-019-00717-0.
- [2] M. M. LaSorda, J. Borky, and R. Sega, “Model-Based Systems Architecting with Decision Quantification for Cybersecurity, Cost, and Performance”, in *2020 IEEE Aerospace Conference*, Mar. 2020, pp. 1–13. DOI: 10.1109/AERO47225.2020.9172283. [Online]. Available: <https://ieeexplore.ieee.org/document/9172283> (visited on Oct. 4, 2023).
- [3] M. Chodas, R. Masterson, and O. de Weck, “Addressing Deep Uncertainty in Space System Development through Model-based Adaptive Design”, in *2020 IEEE Aerospace Conference*, Mar. 2020, pp. 1–17. DOI: 10.1109/AERO47225.2020.9172672. [Online]. Available: <https://ieeexplore.ieee.org/document/9172672> (visited on Oct. 4, 2023).
- [4] M. M. LaSorda, J. Borky, and R. Sega, “Model-based architecture optimization for major acquisition analysis of alternatives”, in *2018 IEEE Aerospace Conference*, Mar. 2018, pp. 1–8. DOI: 10.1109/AERO.2018.8396526. [Online]. Available: <https://ieeexplore.ieee.org/document/8396526> (visited on Oct. 4, 2023).
- [5] B. P. Douglass, “Chapter 5 - Agile Systems Requirements Definition and Analysis”, in *Agile Systems Engineering*, B. P. Douglass, Ed., Boston: Morgan Kaufmann, Jan. 1, 2016, pp. 189–279, ISBN: 978-0-12-802120-0. DOI: 10.1016/B978-0-12-802120-0.00005-9. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B9780128021200000059> (visited on Nov. 15, 2023).
- [6] M. Wäschle, A. Martin, A. Radimersky, *et al.*, “Supporting The Modelling In Mbse By Applying Product Generation Engineering Using Electronic Compact Actuators As An Example”, *Proceedings of the Design Society: DESIGN Conference*, vol. 1, pp. 2425–2434, May 2020, ISSN: 2633-7762. DOI: 10.1017/dsd.2020.293. [Online]. Available: https://www.cambridge.org/core/product/identifier/S2633776220002939/type/journal_article (visited on Apr. 19, 2023).

- [7] S. Friedenthal, A. Moore, and R. Steiner, “An Automobile Example Using the SysML Basic Feature Set”, in *A Practical Guide to SysML*, Elsevier, 2015, pp. 53–81. DOI: 10.1016/b978-0-12-800202-5.00004-7.
- [8] A. Gal, “Ontology Engineering”, in *Encyclopedia of Database Systems*, L. I. N. G. LIU and M. T. ÖZSU, Eds., Boston, MA: Springer US, 2009, pp. 1972–1973, ISBN: 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_1315. [Online]. Available: https://doi.org/10.1007/978-0-387-39940-9_1315 (visited on Apr. 24, 2023).
- [9] O. M. Group, *OMG Unified Modeling Language™ (OMG UML), Superstructure*, Object Modelling Group, Nov. 14, 2014. [Online]. Available: <https://www.omg.org/spec/UML/2.4/Superstructure/PDF> (visited on Aug. 22, 2022).
- [10] P. de Saqui-Sannes, R. Vingerhoeds, C. Garion, and X. Thirioux, “A Taxonomy of MBSE Approaches by Languages, Tools and Methods”, *IEEE Access*, vol. PP, pp. 1–1, Jan. 1, 2022. DOI: 10.1109/ACCESS.2022.3222387.
- [11] OMG, *OMG Systems Modeling Language V1.6*, Jan. 11, 2019. [Online]. Available: <https://sysml.org/.res/docs/specs/OMGSysML-v1.6-19-11-01.pdf> (visited on Nov. 29, 2022).
- [12] de Koning, “SysML Version 2 - Final Stretch”, Nov. 23, 2022. [Online]. Available: <https://indico.esa.int/event/407/contributions/7390/attachments/4793/7804/1710%20-%20Abstract%20-%20SysML%20Version%20%20Final%20Stretch.pdf> (visited on Nov. 29, 2022).
- [13] “OSMoSE - Home”. (), [Online]. Available: https://mb4se.esa.int/OSMOSE_Main.html (visited on Nov. 30, 2022).
- [14] “About the OMG System Modeling Language Specification Version 1.6”. (), [Online]. Available: <https://www.omg.org/spec/SysML/> (visited on Jan. 13, 2022).
- [15] J.-L. Voirin, S. Bonnet, D. Exertier, and V. Normand, “Simplifying (and enriching) SysML to perform functional analysis and model instances”, *INCOSE International Symposium*, vol. 26, no. 1, pp. 253–268, Jul. 2016, ISSN: 23345837. DOI: 10.1002/j.2334-5837.2016.00158.x. [Online]. Available: <https://onlinelibrary.wiley.com/doi/10.1002/j.2334-5837.2016.00158.x> (visited on Oct. 31, 2022).

-
- [16] L. Li, N. L. Soskin, A. Jbara, *et al.*, “Model-Based Systems Engineering for Aircraft Design With Dynamic Landing Constraints Using Object-Process Methodology”, *IEEE Access*, vol. 7, pp. 61 494–61 511, 2019, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2915917.
- [17] M. Di Maio, T. Weilkiens, O. Hussein, *et al.*, “Evaluating MBSE Methodologies Using the FEMMP Framework”, in *2021 IEEE International Symposium on Systems Engineering (ISSE)*, Sep. 2021, pp. 1–8. DOI: 10.1109/ISSE51541.2021.9582465.
- [18] *BSISO/PAS 19450 Automation systems and integration - Object-Process Methodology*, Intenational Standards Organisation, Dec. 15, 2015.
- [19] M. Peleg and D. Dori, “The model multiplicity problem: Experimenting with real-time specification methods”, *IEEE Transactions on Software Engineering*, vol. 26, no. 8, pp. 742–759, Aug. 2000, ISSN: 1939-3520. DOI: 10.1109/32.879812.
- [20] J. (Somekh, M. Choder, and D. Dori, “Conceptual Model-Based Systems Biology: Mapping Knowledge and Discovering Gaps in the mRNA Transcription Cycle”, *PloS one*, vol. 7, e51430, Dec. 20, 2012. DOI: 10.1371/journal.pone.0051430.
- [21] GENSYS. “GENSYS Key Concepts”. (), [Online]. Available: https://vitechcorp.com/resources/GENESYS/onlinehelp/desktop/Key_Concepts.htm (visited on Nov. 9, 2022).
- [22] P. Minacapilli, F. Zurita, S. Campo Perez, *et al.*, “Small satellites mission design enhancement through MBSE and DDSE toolchain”, Nov. 24, 2022.
- [23] S. International, *Architecture Analysis and Design Language (AADL)*, SAE International. [Online]. Available: <https://www.sae.org/standards/content/as5506d/preview/> (visited on Nov. 30, 2022).
- [24] *SLIM Specifaction*, COMPASS Consortium, Aug. 27, 2016. [Online]. Available: <https://www.compass-toolset.org/docs/slim-specification.pdf> (visited on Nov. 20, 2022).
- [25] T. Bayer, J. Day, E. Dodd, *et al.*, “Europa Clipper: MBSE Proving Ground”, in *2021 IEEE Aerospace Conference (50100)*, Mar. 2021, pp. 1–19. DOI: 10.1109/AER050100.2021.9438186.

- [26] Marco Ferrogali, “MBSE at the heart of Airbus Digital Transformation”, Keynote Speech, presented at the MBSE2022 (Toulouse), Oct. 22, 2023. [Online]. Available: <https://indico.esa.int/event/407/contributions/7561/attachments/5009/7810/Keynote%20-%20MBSE%20at%20the%20heart%20of%20the%20Airbus%20Digital%20Transformation.pdf> (visited on Feb. 24, 2023).
- [27] L. Timperley, L. Berthoud, C. Snider, *et al.*, “Towards Improving the Design Space Exploration Process Using Generative Design With MBSE”, in *2023 IEEE Aerospace Conference*, Mar. 2023, pp. 1–20. DOI: 10.1109/AER055745.2023.10116019. [Online]. Available: <https://ieeexplore.ieee.org/document/10116019/metrics#metrics> (visited on Sep. 26, 2023).
- [28] “Model-based System and Architecture Engineering with the Arcadia Method - 1st Edition”. (), [Online]. Available: <https://www.elsevier.com/books/model-based-system-and-architecture-engineering-with-the-arcadia-method/voirin/978-1-78548-169-7> (visited on Mar. 21, 2023).
- [29] “Engineering Systems Design Rhapsody - Details”. (Jun. 16, 2022), [Online]. Available: <https://www.ibm.com/products/systems-design-rhapsody/details> (visited on Mar. 20, 2023).
- [30] OPMCloud. “OPMCloud Features”, OPMCloud. (), [Online]. Available: <https://www.opcloud.tech/features> (visited on Nov. 10, 2022).
- [31] “Overview - TASTE”. (), [Online]. Available: <https://taste.tuxfamily.org/wiki/index.php?title=Overview> (visited on Nov. 29, 2022).
- [32] S. Duncan, “Flight Software Development with TASTE”, p. 4,
- [33] M. Bozzano, H. Bruintjes, A. Cimatti, *et al.*, “COMPASS 3.0”, in *Tools and Algorithms for the Construction and Analysis of Systems*, T. Vojnar and L. Zhang, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2019, pp. 379–385, ISBN: 978-3-030-17462-0. DOI: 10.1007/978-3-030-17462-0_25.
- [34] M. Bozzano, A. Cimatti, J.-P. Katoen, *et al.*, *The COMPASS Approach: Correctness, Modelling and Performability of Aerospace Systems*. Sep. 15, 2009, vol. 5775, p. 186, 173 pp., ISBN: 978-3-642-04467-0. DOI: 10.1007/978-3-642-04468-7_15.

-
- [35] A. Bombardelli, M. Bozzano, R. Cavada, *et al.*, “COMPASTA: Extending TASTE with Formal Design and Verification Functionality”, in *Model-Based Safety and Assessment*, C. Seguin, M. Zeller, and T. Prosvirnova, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2022, pp. 21–27, ISBN: 978-3-031-15842-1. DOI: 10.1007/978-3-031-15842-1_2.
- [36] J.-L. Voirin, “2 - Main Perspectives Structuring the Modeling Approach”, in *Model-Based System and Architecture Engineering with the Arcadia Method*, J.-L. Voirin, Ed., Elsevier, Jan. 1, 2018, pp. 15–18, ISBN: 978-1-78548-169-7. DOI: 10.1016/B978-1-78548-169-7.50002-0. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B9781785481697500020> (visited on Mar. 21, 2023).
- [37] L. Batista and O. Hammami, “Capella based system engineering modelling and multi-objective optimization of avionics systems”, in *2016 IEEE International Symposium on Systems Engineering (ISSE)*, Oct. 2016, pp. 1–8. DOI: 10.1109/SysEng.2016.7753127.
- [38] S. Friedenthal, A. Moore, and R. Steiner, “Residential Security System Example Using the Object-Oriented Systems Engineering Method”, in *A Practical Guide to SysML*, Elsevier, 2015, pp. 417–504. DOI: 10.1016/b978-0-12-800202-5.00017-5.
- [39] Hans-Peter Hoffman, *Model-Based Systems Engineering with Rational Rhapsody and Rational Harmony for Systems Engineering*, Deskbook 3.1.2. May 27, 2022. [Online]. Available: <https://www.ibm.com/support/pages/model-based-systems-engineering-rational-rhapsody-and-rational-harmony-systems-engineering-deskbook-312> (visited on Nov. 17, 2023).
- [40] Hoppe, M. “RePoSyD wiki”. (Oct. 28, 2023), [Online]. Available: <https://reposyd.de/start> (visited on Nov. 16, 2023).
- [41] “IEEE Standard for Application and Management of the Systems Engineering Process”, *IEEE Std 1220-2005 (Revision of IEEE Std 1220-1998)*, pp. 1–96, Sep. 2005. DOI: 10.1109/IEEESTD.2005.96469.
- [42] S. T. M. Beyerlein, “RePoSyD Model Based Systems Engineering: Methodology Assessment Using the FEMMP Framework and a Steam Engine Case Study”, BA thesis, Technische Hochschule Ingolstadt, Oct. 27, 2020.

- [43] D. Long and Z. Scott, *A Primer for Model-Based Systems Engineering*. Lulu.com, 2011, 126 pp., ISBN: 978-1-105-58810-5. Google Books: pCaoAwAAQBAJ.
- [44] J. E. Long, “Relationships Between Common Graphical Representations Used in System Engineering”, *INSIGHT*, vol. 21, no. 1, pp. 8–11, 2018, ISSN: 2156-4868. DOI: 10.1002/inst.12183. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/inst.12183> (visited on Nov. 16, 2022).
- [45] D. Dori, R. Martin, and A. Blekhman, *Model-Based Meta-Standardization*. May 8, 2010, p. 597, 593 pp. DOI: 10.1109/SYSTEMS.2010.5482321.
- [46] “Valispace Home”, Valispace. (), [Online]. Available: <https://www.valispace.com/> (visited on Mar. 21, 2023).
- [47] M. Bandecchi, B. Melton, and B. Gardini, “The ESA/ESTEC concurrent design facility”, Jan. 1, 2000.
- [48] J.-B. Bernaudin, “MBSE on MSR ERO: A use case”, presented at the MBSE2021, Toulouse, Sep. 29, 2021. [Online]. Available: <https://indico.esa.int/event/386/contributions/6227/attachments/4269/6378/1145%20-%20mbse%20on%20msr%20ero%20a%20use%20case.pdf> (visited on Nov. 29, 2021).
- [49] C. Coelho, *MBSE Best Practices Project CGI Executive Summary*, ESA, 2021. [Online]. Available: https://nebula.esa.int/sites/default/files/neb_study/2520/C4000133517ExS.pdf (visited on Nov. 29, 2022).
- [50] J. Gregory, “A Model-Based Framework for Early-Stage Analysis of Spacecraft”, Ph.D. dissertation, University of Bristol, Jul. 29, 2022. [Online]. Available: https://www.researchgate.net/publication/363265053_A_Model-Based_Framework_for_Early-Stage_Analysis_of_Spacecraft (visited on Jan. 12, 2022).
- [51] A. Dalvi, A. Razban, H. El-Mounyari, *et al.*, *Integrated System Model of District Cooling for Energy Consumption Optimization*. Nov. 25, 2020.
- [52] “Open Modelica”. (), [Online]. Available: <https://openmodelica.org/> (visited on Mar. 21, 2023).
- [53] Mathworks. “System Modeling and Simulation - MATLAB & Simulink Solutions”. (), [Online]. Available: <https://uk.mathworks.com/solutions/system-design-simulation.html> (visited on Nov. 10, 2022).

-
- [54] “Collimator - Data driven design and simulation”. (), [Online]. Available: <https://www.collimator.ai/> (visited on Nov. 15, 2022).
- [55] “Cameo Simulation Toolkit - CATIA - Dassault Systèmes®”. (), [Online]. Available: <https://www.3ds.com/products-services/catia/products/no-magic/cameo-simulation-toolkit/> (visited on Mar. 21, 2023).
- [56] D. N. Mavris, K. Griendling, and C. E. Dickerson, “Relational-oriented systems engineering and technology tradeoff analysis framework”, *Journal of Aircraft*, vol. 50, no. 5, pp. 1564–1575, 2013, ISSN: 15333868. DOI: 10.2514/1.C032079.
- [57] F. Peres and M. Castelli, “Combinatorial Optimization Problems and Metaheuristics: Review, Challenges, Design, and Development”, *Applied Sciences*, vol. 11, no. 14, p. 6449, 14 Jan. 2021, ISSN: 2076-3417. DOI: 10.3390/app11146449. [Online]. Available: <https://www.mdpi.com/2076-3417/11/14/6449> (visited on Dec. 1, 2022).
- [58] E. Polak, *Optimization: Algorithms and Consistent Approximations*. Springer Science & Business Media, Dec. 6, 2012, 801 pp., ISBN: 978-1-4612-0663-7. Google Books: uoXuBwAAQBAJ.
- [59] H. K. R. Ong and T. Sortrakul, “Comparison of Selection Methods of Genetic Algorithms for Automated Component-Selection of Design Synthesis with Model-Based Systems Engineering”, *Social Science*, p. 20, 2018.
- [60] S. Paterna, M. Santoni, and L. Bruzzone, “An approach based on multiobjective genetic algorithms to schedule observations in planetary remote sensing missions”, *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 13, pp. 4714–4727, 2020, ISSN: 21511535. DOI: 10.1109/JSTARS.2020.3015284.
- [61] IEEE Staff, “Multi Domain optimization with SysML modeling”, IEEE, 2015, ISBN: 978-1-4673-7929-8.
- [62] J. Mádar, J. Abonyi, and F. Szeifert, “Interactive particle swarm optimization”, in *Proceedings - 5th International Conference on Intelligent Systems Design and Applications 2005, ISDA '05*, vol. 2005, 2005, pp. 314–319, ISBN: 0-7695-2286-6. DOI: 10.1109/ISDA.2005.58.

- [63] G. Wu, H. Wang, W. Pedrycz, *et al.*, “Satellite observation scheduling with a novel adaptive simulated annealing algorithm and a dynamic task clustering strategy”, *Computers and Industrial Engineering*, vol. 113, pp. 576–588, Nov. 1, 2017, ISSN: 03608352. DOI: 10.1016/j.cie.2017.09.050.
- [64] J. Liu, T. Dwyer, G. Tack, *et al.*, “Supporting the Problem-Solving Loop: Designing Highly Interactive Optimisation Systems”, Sep. 7, 2020. arXiv: 2009.03163 [cs]. [Online]. Available: <http://arxiv.org/abs/2009.03163> (visited on Jan. 21, 2022).
- [65] M. Halvorson, J. Fuchs, D. Thomas, *et al.*, “Model-Based Mission Planning: Reducing Mission Planning Costs by Generating Mission-Unique Architecture and Process Frameworks”, presented at the 73rd International Astronautical Congress (IAC), Paris, France: International Astronautical Federation (IAF), Sep. 18, 2022.
- [66] C. R. Hicks, *Fundamental Concepts in the Design of Experiments*. Holt, Rinehart and Winston, 1973, 372 pp., ISBN: 978-0-03-080132-7. Google Books: TuVQAAAAMAAJ.
- [67] L. Caldas, “GENE_ARCH: An Evolution-Based Generative Design System for Sustainable Architecture”, in *Intelligent Computing in Engineering and Architecture*, I. F. C. Smith, Ed., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2006, pp. 109–118, ISBN: 978-3-540-46247-7. DOI: 10.1007/11888598_12.
- [68] J.-P. Ceglarek, R. Boumghar, and R. Bertrand, “AI4CE - Automated Space Mission Design Concepts Generation with Reinforcement Learning”, p. 5,
- [69] S. Meacham, F. Gioulekas, and K. Phalp, “SysML based Design for Variability enabling the Reusability of Legacy Systems towards the support of Diverse Standard Compliant Implementations or Standard Updates: The Case of IEEE-802.15.6 Standard for e-Health Applications”, in *Proceedings of the Eighth EAI International Conference on Simulation Tools and Techniques*, Athens, Greece: ACM, 2015, ISBN: 978-1-63190-079-2. DOI: 10.4108/eai.24-8-2015.2261108. [Online]. Available: <http://eudl.eu/doi/10.4108/eai.24-8-2015.2261108> (visited on Nov. 17, 2023).
- [70] V. Singh and N. Gu, “Towards an integrated generative design framework”, *Design Studies*, vol. 33, no. 2, pp. 185–207, Mar. 1, 2012, ISSN: 0142-694X. DOI: 10.1016/

-
- j.destud.2011.06.001. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0142694X11000391> (visited on Dec. 8, 2022).
- [71] A. Kiryakov, “Ontologies for Knowledge Management”, in *Semantic Web Technologies*, John Wiley & Sons, Ltd, 2006, pp. 115–138, ISBN: 978-0-470-03033-2. DOI: 10.1002/047003033X.ch7. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/047003033X.ch7> (visited on Mar. 21, 2023).
- [72] “Neo4j Graph Data Platform | Graph Database Management System”. (), [Online]. Available: <https://neo4j.com/> (visited on Mar. 16, 2023).
- [73] “Introduction - Cypher Manual”, Neo4j Graph Data Platform. (), [Online]. Available: <https://neo4j.com/docs/cypher-manual/5/introduction/> (visited on Oct. 4, 2023).
- [74] F. Schummer and M. Hyba, “An approach for system analysis with model-based systems engineering and graph data engineering”, *Data-Centric Engineering*, vol. 3, e33, 2022, ISSN: 2632-6736. DOI: 10.1017/dce.2022.33. [Online]. Available: https://www.cambridge.org/core/product/identifier/S2632673622000338/type/journal_article (visited on Sep. 15, 2023).
- [75] L. Ehrlinger and W. Wöß, “Towards a Definition of Knowledge Graphs”,
- [76] C. Feilmayr and W. Wöß, “An analysis of ontologies and their success factors for application to business”, *Data & Knowledge Engineering*, vol. 101, pp. 1–23, Jan. 1, 2016, ISSN: 0169-023X. DOI: 10.1016/j.datak.2015.11.003. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0169023X1500110X> (visited on Mar. 21, 2023).
- [77] E. W. Dijkstra, “A note on two problems in connexion with graphs”, *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, Dec. 1, 1959, ISSN: 0945-3245. DOI: 10.1007/BF01386390. [Online]. Available: <https://doi.org/10.1007/BF01386390> (visited on Mar. 27, 2023).
- [78] P. E. Hart, N. J. Nilsson, and B. Raphael, “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”, *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, Jul. 1968, ISSN: 2168-2887. DOI: 10.1109/TSSC.1968.300136.

- [79] L. Timperley, *Ghipag/MBSE_Design_Space_Problem_Space*, Oct. 2, 2023. [Online]. Available: https://github.com/Ghipag/MBSE_Design_Space_Problem_Space (visited on Oct. 4, 2023).
- [80] A. K. Raz, C. R. Kenley, and D. A. DeLaurentis, “System architecting and design space characterization”, *Systems Engineering*, vol. 21, no. 3, pp. 227–242, May 1, 2018, ISSN: 15206858. DOI: 10.1002/sys.21439.
- [81] “ISO/IEC/IEEE International Standard - Systems and software engineering—System life cycle processes”, *ISO/IEC/IEEE 15288:2023(E)*, pp. 1–128, May 2023. DOI: 10.1109/IEEESTD.2023.10123367. [Online]. Available: <https://ieeexplore.ieee.org/document/10123367> (visited on Jul. 2, 2024).
- [82] “ISO/IEC/IEEE International Standard - Systems and software engineering – Vocabulary”, *ISO/IEC/IEEE 24765:2010(E)*, pp. 1–418, Dec. 2010. DOI: 10.1109/IEEESTD.2010.5733835. [Online]. Available: <https://ieeexplore.ieee.org/document/5733835> (visited on Oct. 27, 2023).
- [83] G. S. Parnell, Ed., *Trade-Off Analytics: Creating and Exploring the System Tradespace*, 1st edition. Hoboken, New Jersey: Wiley, Jan. 6, 2017, 640 pp., ISBN: 978-1-119-23753-2.
- [84] M. Maier, *The Art of Systems Architecting*. Jan. 6, 2009, ISBN: 978-0-429-19614-0. DOI: 10.1201/9781420079142.
- [85] J. M. Stecklein, J. Dabney, B. Dick, *et al.*, “Error Cost Escalation Through the Project Life Cycle”, presented at the 14th Annual International Symposium, Toulouse, Jun. 19, 2004. [Online]. Available: <https://ntrs.nasa.gov/citations/20100036670> (visited on Jul. 5, 2024).
- [86] B. W. Boehm, *Software Engineering Economics*. Prentice-Hall, 1981, 806 pp., ISBN: 978-0-13-822122-5. Google Books: VphQAAAAMAAJ.
- [87] A. M. Madni, “Novel Options Generation”, in *Transdisciplinary Systems Engineering: Exploiting Convergence in a Hyper-Connected World*, A. M. Madni, Ed., Cham: Springer International Publishing, 2018, pp. 89–102, ISBN: 978-3-319-62184-5. DOI: 10.1007/978-3-319-62184-5_6. [Online]. Available: https://doi.org/10.1007/978-3-319-62184-5_6 (visited on Jul. 5, 2024).
- [88] J. Bussemaker, P. D. Ciampa, and B. Nagel, *SYSTEM ARCHITECTURE DESIGN SPACE MODELING AND OPTIMIZATION ELEMENTS*. Nov. 6, 2021.

-
- [89] J.-L. Voirin, “ARCADIA REFERENCE: WORKFLOW AND ACTIVITIES”, 2023.
- [90] J. Bussemaker, L. Boggero, and P. D. Ciampa, “From System Architecting to System Design and Optimization: A Link Between MBSE and MDAO”, *INCOSE International Symposium*, vol. 32, pp. 343–359, Sep. 26, 2022. DOI: 10.1002/iis2.12935.
- [91] D. M. Judt and C. P. Lawson, “Development of an automated aircraft subsystem architecture generation and analysis tool”, Dec. 1, 2015, ISSN: 0264-4401. DOI: 10.1108/EC-02-2014-0033. [Online]. Available: <https://dspace.lib.cranfield.ac.uk/handle/1826/10838> (visited on Jun. 11, 2024).
- [92] D. Herber, J. Allison, R. Buettner, *et al.*, *Architecture Generation and Performance Evaluation of Aircraft Thermal Management Systems Through Graph-based Techniques*. Jan. 6, 2020. DOI: 10.2514/6.2020-0159.
- [93] D. Mavris, C. de Tenorio, and M. Armstrong, “Methodology for Aircraft System Architecture Definition”, in *46th AIAA Aerospace Sciences Meeting and Exhibit*, American Institute of Aeronautics and Astronautics. DOI: 10.2514/6.2008-149. [Online]. Available: <https://arc.aiaa.org/doi/abs/10.2514/6.2008-149> (visited on Jun. 10, 2024).
- [94] W. L. Simmons, “A framework for decision support in systems architecting”, Thesis, Massachusetts Institute of Technology, 2008. [Online]. Available: <https://dspace.mit.edu/handle/1721.1/42912> (visited on Jun. 10, 2024).
- [95] R. H. Madeira, D. H. de Sousa Pinto, and M. Forlingieri, “Variability on System Architecture using Airbus MBPLE for MOFLT Framework”, *INCOSE International Symposium*, vol. 33, no. 1, pp. 601–615, 2023, ISSN: 2334-5837. DOI: 10.1002/iis2.13041. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/iis2.13041> (visited on May 22, 2024).
- [96] J. Bussemaker, P. D. Ciampa, and B. Nagel, *System Architecture Design Space Exploration: An Approach to Modeling and Optimization*. Jun. 15, 2020. DOI: 10.2514/6.2020-3172.
- [97] W. J. von Eschenbach, “Transparency and the Black Box Problem: Why We Do Not Trust AI”, *Philosophy & Technology*, vol. 34, no. 4, pp. 1607–1622, Dec. 1, 2021, ISSN: 2210-5441. DOI: 10.1007/s13347-021-00477-0. [Online]. Available: <https://doi.org/10.1007/s13347-021-00477-0> (visited on Jul. 8, 2024).

- [98] “System Architecture: Strategy and Product Development for Complex Systems, First Edition [Book]”. (), [Online]. Available: <https://www.oreilly.com/library/view/system-architecture-strategy/9780136462989/> (visited on Jul. 30, 2024).
- [99] D. Selva, B. Cameron, and E. Crawley, “Patterns in System Architecture Decisions: PATTERNS IN SYSTEM ARCHITECTURE DECISIONS”, *Systems Engineering*, vol. 19, no. 6, pp. 477–497, Nov. 2016, ISSN: 10981241. DOI: 10.1002/sys.21370. [Online]. Available: <https://onlinelibrary.wiley.com/doi/10.1002/sys.21370> (visited on Jul. 30, 2024).
- [100] J.-L. Voirin, “ARCADIA LANGUAGE REFERENCE: META MODEL”, 2023.
- [101] D. K. Hitchins, *Systems Engineering: A 21st Century Systems Methodology*. John Wiley & Sons, Mar. 11, 2008, 533 pp., ISBN: 978-0-470-51875-5. Google Books: [tdZod1zaIeQC](#).
- [102] C. S. Wasson, “3.3.1 System Phases, Modes, and States: Solutions to Controversial Issues”, *INCOSE International Symposium*, vol. 21, no. 1, pp. 279–294, Jun. 2011, ISSN: 2334-5837, 2334-5837. DOI: 10.1002/j.2334-5837.2011.tb01205.x. [Online]. Available: <https://incose.onlinelibrary.wiley.com/doi/10.1002/j.2334-5837.2011.tb01205.x> (visited on Aug. 13, 2024).