CRANFIELD UNIVERSITY

Yogesh Hanumant Bile

Component-driven Computational Design of Complex Engineering Systems

Supervisors: Prof. Marin D. Guenov; Dr A. Molina-Cristobal

PhD

Academic Year: 2017-2018

Department of Aerospace Engineering

School of Aerospace, Transport and Manufacturing

August 2018

CRANFIELD UNIVERSITY

School of Aerospace, Transport and Manufacturing

PhD

Academic Year: 2017-2018

Yogesh Hanumant Bile

Component-driven Computational Design of Complex Engineering Systems

Supervisor: Prof. Marin D. Guenov, Dr A. Molina-Cristóbal

April 2018

Declaration of Authorship

I, Yogesh Hanumant Bile, declare that this thesis titled, 'Component-driven Computational Design of Complex Engineering Systems' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

“Dedicated to the memory of my mother”

Abstract

During the conceptual design of complex systems, architects study a number of different options, which comprise the architectural design space. Usually, new system architectures (SAs) are created by modifying existing ones, e.g., by deleting existing and/or adding new elements. Once the concept is synthesised, the architect wishes to swiftly find the effect of the proposed architectural changes at system level. This would involve sizing of the modified sub-systems, and then, obtaining the system level performance. In turn, this involves time-consuming activities, such as re-arrangement (orchestration) of computational tasks and models. Also, depending on the results, the architect may undertake further modifications. When doing this, a means to navigate across the RFLP (Requirements-Functional-Logical-Physical) views of the SA may be required, in order to trace elements affected by these modifications. Generally, several iterations are involved between architecting and sizing during conceptual design, which, if manually performed, result in a tedious and time-consuming process. There are existing methods which address this problem, but these have significant limitations in that they are usually system specific and often involve an excessive amount of time-consuming manual tasks.

Within this context, the research aim is to improve the efficiency of the architectural design space exploration (ADSE) process, by automating repetitive computational tasks, thus enabling the designer to swiftly and interactively explore multiple SA options.

A novel method, comprised of two parts, has been developed to achieve the aim. In the first part, a graph-theoretic approach is employed to enable architectural element dependency analysis. Here, the relationships between the architectural elements are stored as a graph. Algorithms, such as 'Depth First Search' and 'Transitive Closure' are

then applied to assist the architect in tracing the dependencies between elements that might be affected by a proposed change to other elements of the SA.

In the second part, the architecture is assessed to find the system level performance. The inputs needed for rapid assessment include the functional and logical views of the SA, and the requisite steady-state computational models associated with each of the 'logical' components. The assessment process itself consists of three steps. In the first step, the sequence of the sub-systems is automatically generated by extracting a sub-systems source-sink 'Dependency Structure Matrix (DSM)' from the logical view, followed by the application of an algorithm which determines the systems' sizing sequence. In the second step, the individual sub-systems and system level workflows are constructed. Here, the computational workflow (a network of computational models) is represented as a bipartite graph. A maximum matching enumeration algorithm is used to find all possible workflows for a given model set, and another algorithm, to choose from these the most computationally efficient one, i.e., the workflow with the lowest number of reversed variables. In the third step, the workflows produced in the second step and sub-systems' sizing sequence obtained in the first step are combined to produce a complete workflow.

To demonstrate and evaluate the proposed enablers, the author developed a prototype object-oriented architecting tool. The enablers were individually and collectively verified on representative test-cases. Comparison with the existing methods confirmed the claimed advantages of the proposed approach, namely, reducing the number of manual activities, which results in swifter and interactive ADSE process. Feedback obtained from experts in the aircraft industry during an initial qualitative evaluation session confirmed the usefulness of the proposed method.

Acknowledgements

This has been a long but enjoyable journey and there are many people to thank. Firstly, I would like to express my sincere gratitude to my supervisors Professor Marin D. Guenov and Dr Arturo Molina-Cristóbal for their guidance and patience throughout this research, and for giving me the opportunity to work alongside them. Their guidance helped me in all the time of research and writing of this thesis.

I would like to thank the following past and present colleagues from our research group for their contribution: Many thanks to Atif Riaz for his continual support, guidance, and encouragement throughout the PhD. I would also like to thank Xin Chen, Stevan van Heerden, and Soufiane El Fassi for their support and feedback during the research. Thank you, Gabriele Luigi Mura and Sergio Jimeno Altelarrea for reading my thesis and giving valuable feedback. I would also like to thank previous members of my group Vitaly Voloshin and Varun Datta for their critical suggestions on my research during initial days of the PhD. Thank you all for your friendship. It was a great time working with you, and I really enjoyed our discussions on technical and non-technical subjects which helped me to enrich my knowledge.

I am thankful to the European Project - TOICA and the centre for Aeronautics for funding my PhD research at Cranfield University.

I also need to extend a special thank you to the design specialists from Airbus UK who helped me to evaluate my research.

I take this opportunity to express gratitude to all the researchers in my department, Emma Turner from Cranfield University library and also the departmental secretaries,

Diane and Elizabeth for helping me at different points during my stay at Cranfield University.

My sincere thanks to my first manager, Aditya Gondhalekar for supporting and guiding me from the start of my professional career. I deeply appreciate your valuable time you have given to provide me with invaluable advice.

Last but not least, I would like to thank my family: my father, my sister and her family, and my wife for supporting and encouraging me throughout my academic career. I would not have come this far without your love, support and confidence in me.

Contents

List of Figures

List of Tables

Abbreviations

AA	**A**rchitecture **A**ssessment
AADL	**A**rchitecture **A**nalysis and **D**esign **L**anguage
AD	**A**rchitecture **D**efinition
AD	**A**xiomatic **D**esign
ADSE	**A**rchitectural **D**esign **S**pace **E**xploration
APU	**A**uxiliary **P**ower **U**nit
ATC	**A**nalytical **T**arget **C**ascading
BG	**B**ipartite **G**raph
CAB	**C**abin
CO	**C**ollaborative **O**ptimisation
CPN	**C**oloured **P**etri **N**et
CV	**C**omputational **V**iew
DRM	**D**esign **R**esearch **M**ethodology
DS	**D**escriptive **S**tudy
DSM	**D**esign **S**tructure **M**atrix (or **D**ependency **S**tructure **M**atrix)
E	**E**nergy
ECS	**E**nvironmental **C**ontrol **S**ystem
ENG	**E**ngine

EPS	**E**lectrical **P**ower **S**ystem
F	**F**unction
FAST	**F**unctional **A**nalysis **S**ystem **T**echnique
FBS	**F**unction **B**ehaviour **S**tructure
FFV	**F**unctional **F**low **V**iew
FHV	**F**unctional **H**ierarchy **V**iew
FPI	**F**ixed **P**oint **I**teration
GUI	**G**raphical **U**ser **I**nterface
HPS	**H**ydraulic **P**ower **S**ystem
I	**I**nformation
IPS	**I**ce **P**rotection **S**ystem
L	**L**ogical
LFV	**L**ogical **F**low **V**iew
LG	**L**anding **G**ear
LHV	**L**ogical **H**ierarchy **V**iew
LTM	**L**ower **T**riangular **M**atrix
M	**M**aterial
MBSE	**M**odel **B**ased **S**ystems **E**ngineering
OMG	**O**bject **M**anagement **G**roup
P	**P**hysical
PN	**P**rime **N**umber
PPS	**P**neumatic **P**ower **S**ystem
PS	**P**rescriptive **S**tudy
R	**R**equirement
RV	**R**equirement **V**iew
SCC	**S**trongly **C**onnected **C**omponent
SE	**S**ystems **E**ngineering
SysML	**S**ystems **M**odelling **L**anguage
TOICA	**T**hermal **O**verall **I**ntegrated **C**onception of **A**ircraft
UML	**U**nified **M**odelling **L**anguage

Symbols

$TNvar$	Total number of variables
$NIvar$	Number of independent variables
$Noutmod$	Sum of the total number of outputs of models
$f(x)$	Optimisation objective function
$g_i(x)$ and $h_i(x)$	Constraint functions
$\circ\!\!-$	Mapping relation
$\twoheadrightarrow$	Derived relation
$\leftarrowtail$	Decomposition relation
Φ	Functions set
Σ	Solutions set
E	Edge set
BG	Bipartite graph
ρ_i	i^{th} requirement
φi	i^{th} function
σ_i	i^{th} solution
μ_i	i^{th} model
$\in$	Element of
$\mathscr{R}$	Relations set

G^i	Initial relations graph
G^{TC}	Transitive closure graph
S	System
SS	Sub-system
C	Solution or component
$\mathcal{U}^i_j$	Input port parameters of j^{th} component of i^{th} sub-system
$\mathcal{P}^i_j$	Component parameters of j^{th} component of i^{th} sub-system
$\mathcal{Y}^i_j$	Output port parameters of j^{th} component of i^{th} sub-system
$\mathcal{M}^i_j$	Models associated with j^{th} component of i^{th} sub-system
$\dot{m}$	Mass flow rate
P	Pressure
T	Temperature
$\mathcal{T}^i$	Targets on i^{th} sub-system
$\forall$	For all
$\exists$	There exists
$\oplus$	Exclusive or
$\subset$	Subset of
$\mathbb{K}$	Known variables
$\mathbb{U}$	Unknown variables
$\emptyset$	Empty set
UDG	Undirected graph
UDG_d	Undirected graph after adding duplicate nodes
$UDG^+_d(e)$	Graph obtained by deleting edge 'e' and its associated nodes
$UDG^-_d(e)$	Graph obtained by deleting only edge 'e'
$P(x)$	Predicate
$\Longleftrightarrow$	Double implication
$\neg$	Negation
$\Longrightarrow$	Implies
$\sum$	Summation Operator
$\wedge$	And

CHAPTER 1

Introduction

An architectural change during conceptual design is not an unusual activity. During this process, there is a need to rapidly understand the effect of the changes on the system level performance to decide on further modifications. Thorough architectural design space exploration in this early design stage is important as these changes can be very expensive when discovered during the later design stages. Finding the effect of the architectural changes involves sizing the architecture. Currently, available (sizing) methods are either system specific or manual, and there is still scope for automation or assistance to the user during the involved computational non-creative tasks.

This PhD work has been undertaken to investigate how to speed up the activities during early stage design. As a starting point, this chapter offers an introduction to the context and scope of the research, problem statement, aim and objectives, and thesis structure.

1.1 Research Context: Conceptual Stage System Design

The design of a complex system, such as an aircraft, consists of three consecutive, iterative stages: conceptual, embodiment and detailed design. These stages[1], in line with the VDI Guidelines 2221[2] and 2222[3], are shown in Figure 1.1. The sub-steps involved in the conceptual design, as proposed by Pahl and Beitz[1], are shown in Figure 1.2. The process starts with taking a list of requirements or design specifications derived from the customer needs. The goal of this stage is to produce a principle solution or concept (which is referred to a system architecture, in this thesis) of the system to be built.

The first step is to identify the essential problems through abstraction, analysing the requirements list, to come up with a solution-neutral overall function of the system or product to be designed. This overall function is decomposed into sub-functions to establish function structures in the second step. The function structure describes the inter-relations among the sub-functions of the overall function. In this report, the process of decomposing one level function into the sub-functions is also referred to as functional reasoning[4][5][6]. Then, the *working principles* that fulfil the sub-functions are searched . A *working principle* is a component or solution that fulfils the function, i.e. the component produces a behaviour that matches the sub-function. All the *working principles*, fulfilling the sub-functions of the overall function, are assembled into a working structure. It is possible to come up with several different working structures of the system by combing various *working principles*. Next, the suitable combinations are selected for further detailing i.e. to firm up the structures into principle solution variants. Available options are assessed against technical and economic criteria to choose the best in class concepts. The selection, as suggested by Pahl and Beitz[1], involves two steps: elimination and preference. In the elimination step, unsuitable solution options are eliminated. The preference stage is executed if there are still several solutions remaining, where the latter are ranked as per their goodness. Only these remaining concepts are evaluated at the end of this stage to pick the principle solution. The evaluation

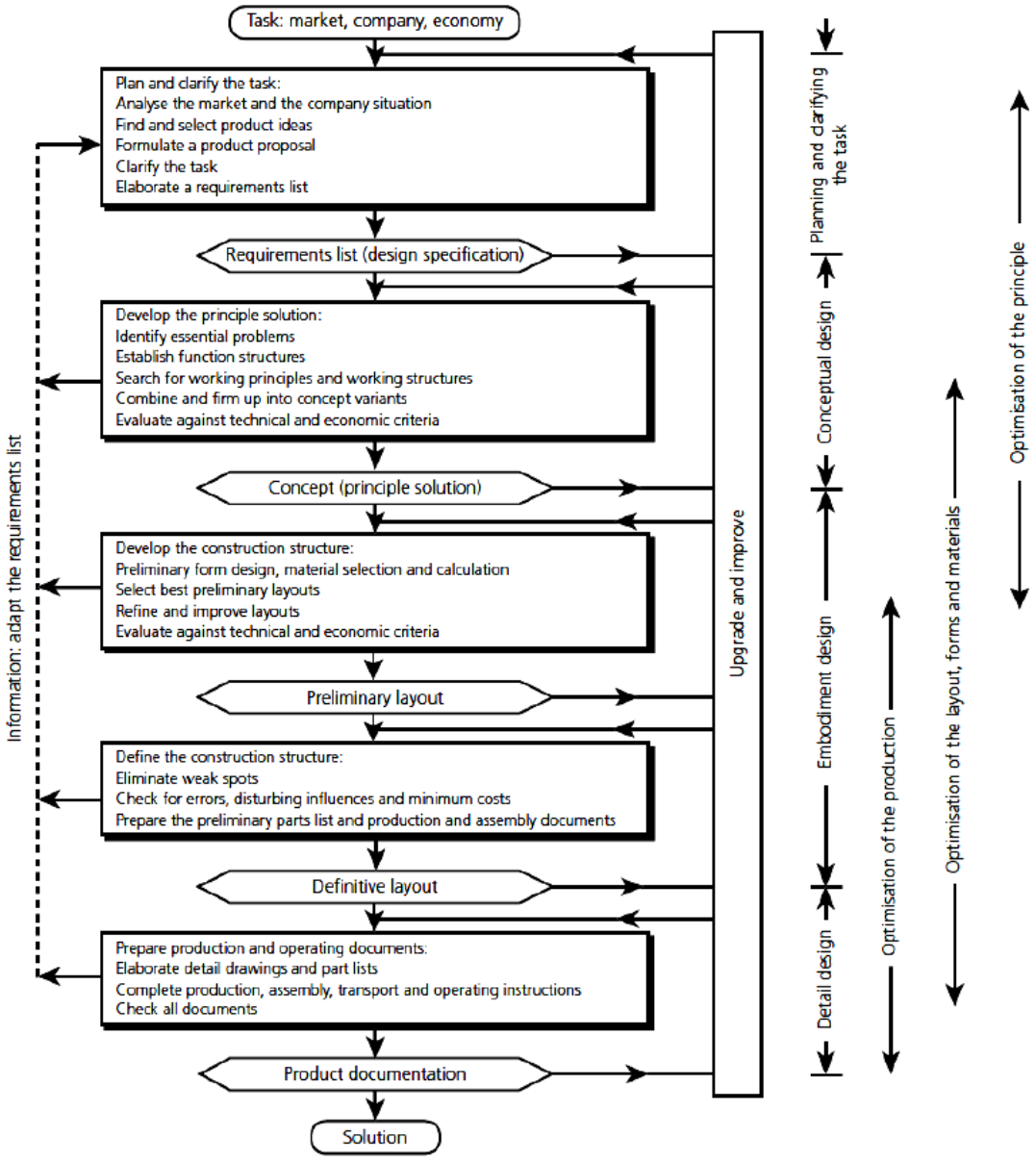


Figure 1.1: *The engineering design process model by Pahl and Beitz*[1]

involves assessment of the concepts against technical, safety, environmental and economic values. The concept or principle variant that scores better than the other variants is chosen and passed on to the next design stage, i.e. embodiment design.

In the embodiment design, an overall layout (construction structure) of the system is determined. In the last phase, detailed design, the arrangement, forms, dimensions and surface properties of all individual parts of the system are laid down. In addition, this phase involves the specification of materials, assessment of production possibilities and

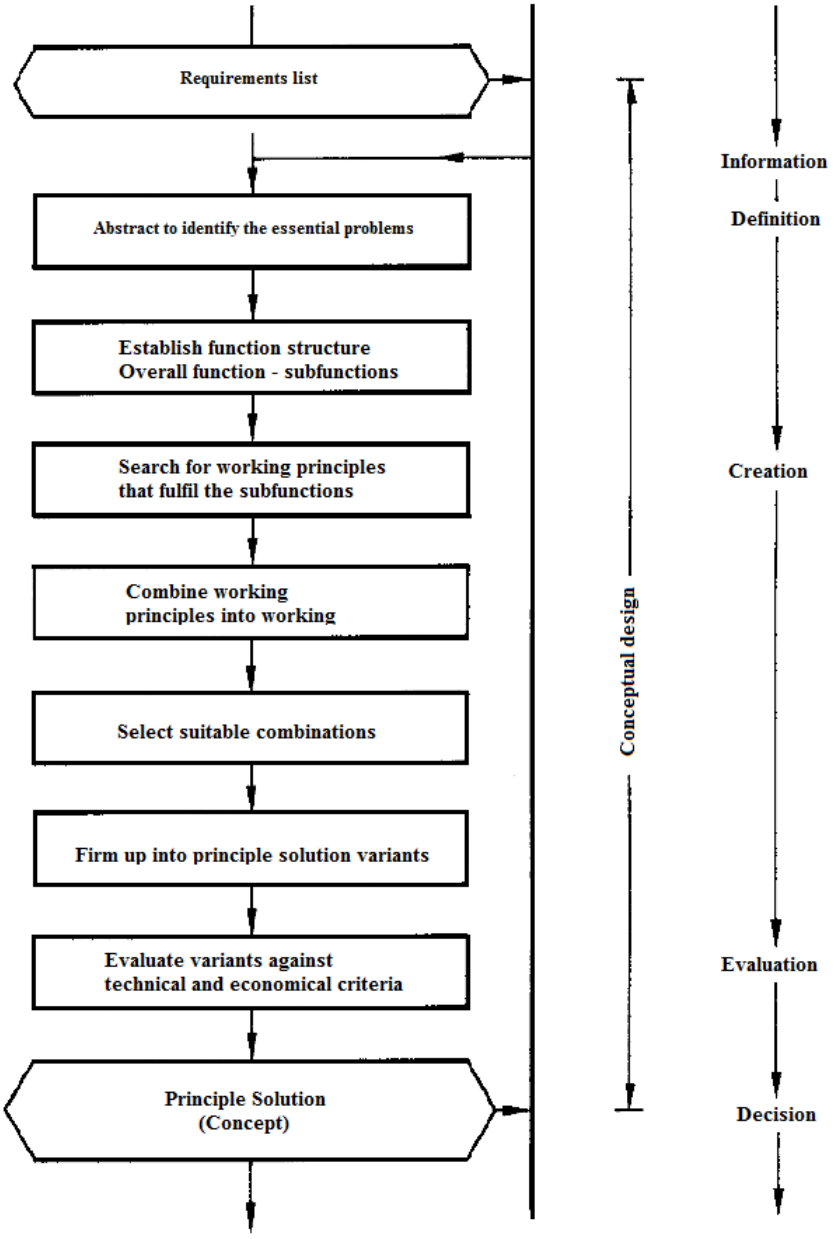


Figure 1.2: *Conceptual design steps*[1].

estimation of cost, and produces design drawings as well as other production documents.

In this research, the term system concept will be referred to as system architecture. The architecture is a collection of different views or perspectives of the system under consideration[7], for example, requirement (R), function (F), logical (L), physical (P) views of the system. There is an analogy between the above views and Pahl and Beitz's function structure and working structure[1]. Here, the functional view (F) is equivalent to

function structures of the system, whereas, logical view (L) is equivalent to the working structure of the system. More details on the terminology are given in the next chapter.

During each of the design stages, several options are generated, which is called 'divergence' in the design; from these options, promising ones are selected and passed onto the next phase, called 'convergence' in the conceptual design phase, as shown in Figure 1.3. This convergence and divergence in the design are performed to explore the design space. In the case of conceptual design, several architectural options for the system to be designed are explored; hence, in this report, the exploration will be referred to as Architectural Design Space Exploration (ADSE).

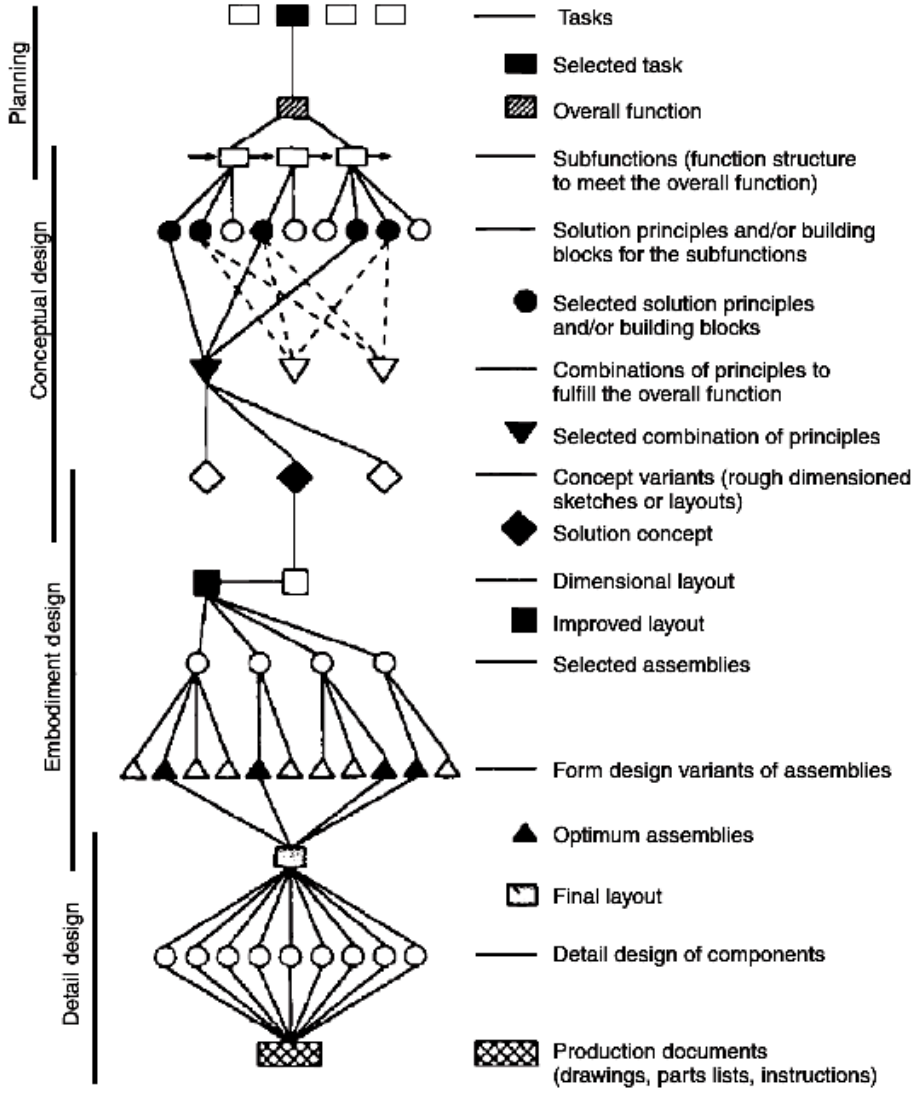


Figure 1.3: *Divergence and convergence in the design (VDI 2222)*[3].

1.2 Research Topic: Architectural Design Space Exploration (ADSE)

The architectural design space exploration contains mainly two activities: 1) architecture definition (or architecting), where the system architecture is defined and 2) architecture assessment, where the system architecture is assessed for its performance. These basic activities are iteratively performed to explore the different system architecture options during ADSE.

1.2.1 Architecture Definition

In the presented research, the system architecture is defined as a collection of the different views or perspectives of the system, and the process of creating these views is called 'architecting' or architecture definition[7]. Usually, these views, such as requirements (R), functions (F), logical (L), physical (P) views of the system, are created by an architect. Often, new architectures are created by infusing new technologies into existing architectures. These existing architectures are referred to as 'baselines' in this thesis. This involves modification of views of the architecture, and the modifications contain addition/deletion of architectural elements or connections between them. Here, the term 'architectural elements' refers to requirements (of R domain), functions (of F domain), solutions/components* (of L domain), and their associated models and parameters (computational domain, for more details see Chapter 5 and Appendix C).

1.2.2 Architecture Assessment

The architecture assessment is defined as the process of determining the system performance for a given architecture. It usually requires, firstly, sizing the system for the architecture and then finding system-level performance[8][9][10][11]. The system sizing

*It should be noted that terms, 'solution' and 'component', are used interchangeably to describe elements of logical (L) domain, unless 'component' refers to a level in the system decomposition (hierarchy).

task is usually decomposed into smaller tasks to deal with the complexity of the system. Then, the sub-tasks are sequenced according to the inter-dependency among them. Next, for each sub-task, a computational sizing workflow (a network of computational models) is created. The workflow is executed to find the sizes of the associated system components. Finally, the system level performance is determined, creating and executing a system level computational workflow. The performance results are employed to decide on further changes in the architecture and eliminate the architecture options that do not meet the requirements or are over-designed (i.e. exceed the requirements by an unnecessary margin).

1.3 Motivation

1.3.1 Importance of Conceptual Design

Conceptual design is a crucial and important part of the system design process as it commits around 80% of the product life-cycle cost[12]. In this stage, the design is flexible and a design change is not as costly as during the embodiment or detail design. It is important to choose a promising architecture option in this phase because it is extremely difficult or impossible to correct fundamental shortcomings of the selected architecture in the subsequent embodiment and detail design phases[1]. Therefore, it is necessary to thoroughly explore the architectural design space before selecting the architecture to be passed onto to the next design stages.

1.3.2 Needs of Architects in Industry with Respect to ADSE

Needs of the architects relevant to the research topic were elicited during the FP7 European Project, Thermal Overall Integrated Conception of Aircraft (TOICA)[13]. A brief summary is presented below.

During the ADSE process, usually, the existing architectures are modified to create a new architecture, and this is rarely a linear process i.e. the views are not created in

a particular order. The architect, in the course of the architecting, is often required to reiterate and navigate between these views in any direction as described by the architects during the interviews. That is, the architect needs to navigate in any direction, for example forward (R → F → L → P) or backward (P → L → F → R), between the different views. Furthermore, the architects also want to trace dependency between architectural elements. This is necessary for well-informed changes in the architecture.

During the modification process, the architect wishes to swiftly find the effect of the architectural changes on the system level performance i.e. to assess the architecture, to decide on further modifications. This, as mentioned in Section 1.2.2, involves sizing of the modified sub-systems and then obtaining the system level performance. In current industrial practice, the architects request to the simulation specialist to conduct above design (sizing) activities for them. However, as mentioned during the interviews[13], the architect wish to be able to, 1) create/modify/edit models for rapid sizing studies, 2) play (try different models) with the involved models of the design studies, and 3) interact with the design parameters to change their values.

1.3.3 Lack of Computational Support in the ADSE

In ADSE, architecture definition and assessment activities are performed repetitively as a result of frequent modifications during the conceptual design stage. A literature review of relevant existing methods (discussed in detail in Chapter-3) has identified that the existing methods lack computational support during the ADSE.

On the architecture definition side, existing architecting methods/tools/languages do not support dependency analysis of architectural elements. On the assessment side, re-arrangement (orchestration) of computational tasks and models is required to size and find system level performance and usually involves humans in the process. Performing these activities manually and repetitively is tedious and time consuming process.

1.4 Aim and Objectives

Considering the needs discussed previously the following aim and objectives are specified.

Aim: *Improve the efficiency of the architectural design space exploration process by increasing automation, enabling the human designer/architect to swiftly and interactively explore multiple system architecture options.*

Objectives:

1. Develop a method to enable the architect to interactively navigate between different architectural views and trace the effect of architectural changes.
2. Automate the architecture assessment process at early design stage.
3. Develop a prototype computational framework for exploring architectural design-space. This is to be used for testing and evaluating the proposed methods.

1.5 PhD Research Methodology

In this research, a design research methodology (DRM)[14] suggested by Blessing and Chakrabarti is adopted. The DRM framework consists of four stages as shown in Figure 1.4. The figure shows the basic means (or inputs) used in and main outcomes (or outputs) produced by each stage, as well as links between these stages. Here, the main process flow is indicated by bold arrows whereas the iterations between the stages are denoted by the light arrows.

Within this framework, there are seven types of research depending on the stages involved and the rigour with which the stages are executed, as shown in Figure 1.5. Considering the research problem, and the available time and resources, the type of the PhD research undertaken was of type-3. Here, the first two stages i.e. Research Clarification (RC) and Descriptive Study I (DS-I) are review-based, that is, only the literature

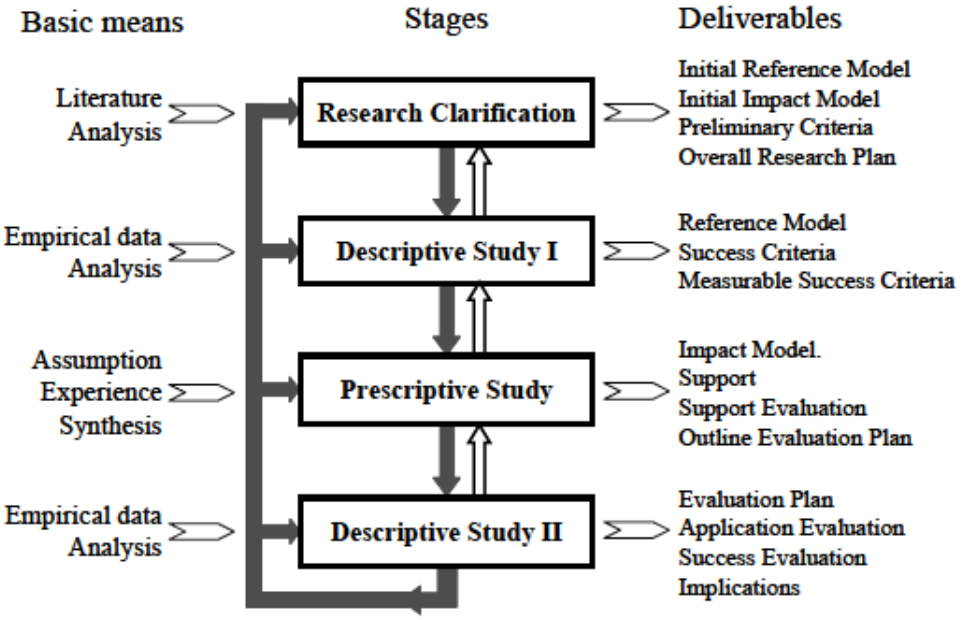


Figure 1.4: *DRM framework: stages, inputs and outputs*[14].

is reviewed; whereas the third stage (Prescriptive Study (PS)) involves comprehensive study. The comprehensive study, in addition to the literature review, contains developing a support. The final stage: DS-II is an initial evaluation of the research.

Research Clarification	Descriptive Study I	Prescriptive Study	Descriptive Study II
1. Review-based	Comprehensive		
2. Review-based	Comprehensive	Initial	
3. Review-based	Review-based	Comprehensive	Initial
4. Review-based	Review-based	Review-based Initial/ Comprehensive	Comprehensive
5. Review-based	Comprehensive	Comprehensive	Initial
6. Review-based	Review-based	Comprehensive	Comprehensive
7. Review-based	Comprehensive	Comprehensive	Comprehensive

Figure 1.5: *Types of design research projects*[14].

The first stage produces an initial reference model (describes the existing situation in the targeted research area) and an impact model (describes the desired situation in the area)†, and preliminary criteria that could be used to measure the success of the research. These outcomes are further refined in DS-1 as the understanding of the research area

†More details on the reference model and impact model notations are given in Appendix B

increases. In this stage, the research success criteria are identified along with measurable success criteria. In the PS stage, increased understanding of the existing situation from previous steps is employed to correct or elaborate the initial impact model describing the desired situation. Accordingly, a support is developed and evaluated. The term - 'support' refers to methods or enablers developed in this research. Finally, the research is evaluated against the success and measurable success criteria identified in DS-I (described in Section 3.5).

The mapping of DRM stages with the thesis chapters describing the outputs from the stages is shown in Figure 1.6.

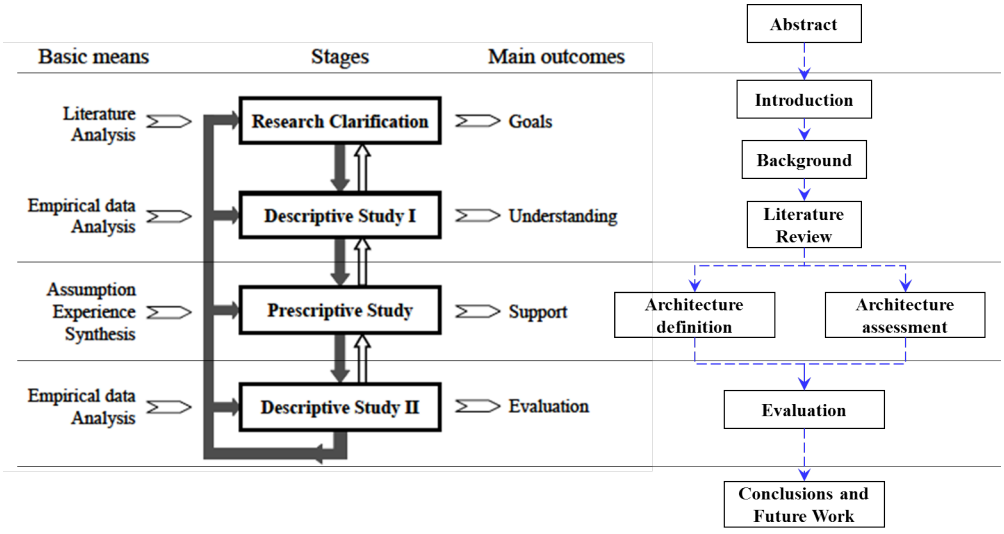


Figure 1.6: *Mapping between the PhD research methodology[14] and thesis structure.*

1.6 Thesis Structure

This thesis is organised in seven chapters as shown in Figure 1.7. This chapter outlined the research problem, aim and objectives, along with a brief description of the context and the research methodology.

The second chapter presents background material relating to computational architecting and sizing. Also, it introduces the basic terminology which is widely used in this work.

In an attempt to refine the context of the research, Chapter 3 critically reviews the current literature on relevant computational architecting and sizing methods. In doing so, it

identifies limitations of existing methods and tools in the current work context and also, the methods which have the potential to be employed in the current work, without/with alterations.

The enablers developed for interactive architecting are described in Chapter 4. They include a technique to enable architectural element dependency analysis and data structures to describe the architectures.

Chapter 5, presents a method for sizing the architecture. It is mainly composed of enablers that support the architecture assessment process.

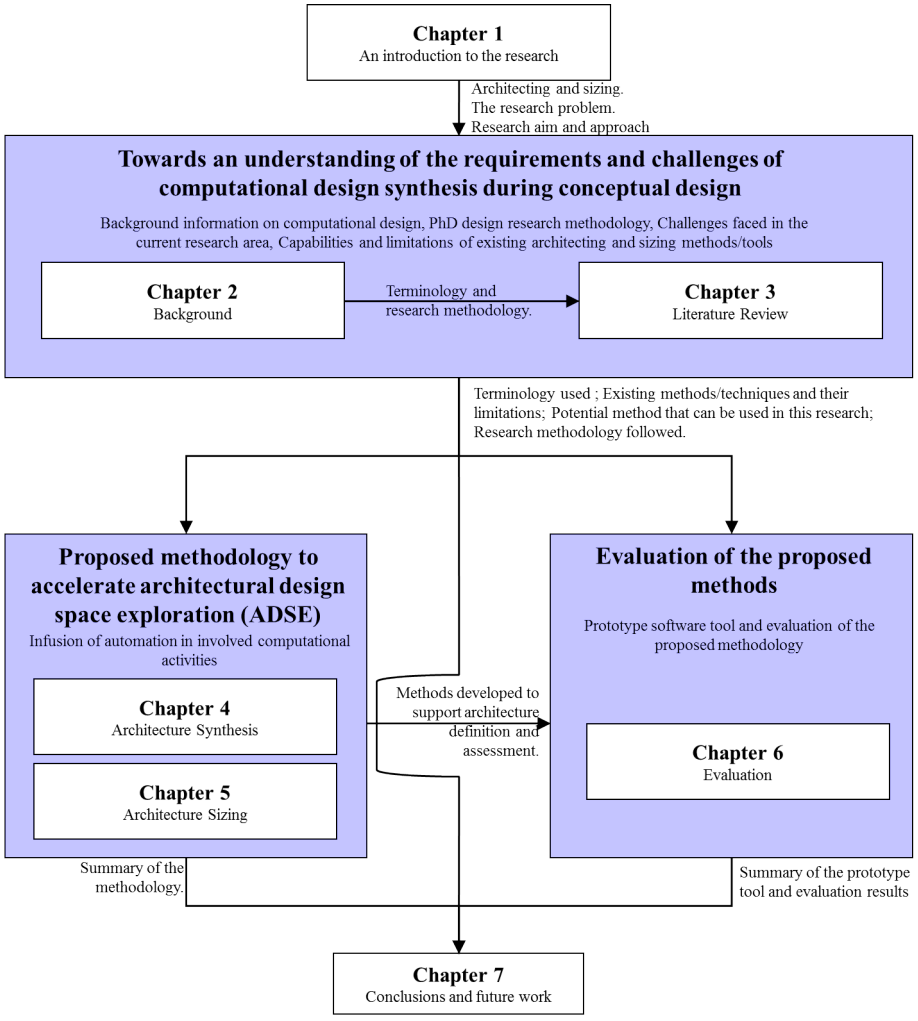


Figure 1.7: *Outline of the thesis structure.*

Chapter 6 describes a prototype object-oriented architecting tool which has been developed in C# for evaluating the research from three types of evaluations, support, application and success, conducted for the research.

Finally, Chapter 7 concludes the research, where, the research contributions to knowledge and limitations of the proposed method are underlined, and the areas that would benefit from further research are set out.

CHAPTER 2

Background

The focus of the research is to develop a support* to assist the user during conceptual design of a complex system. Having introduced the research challenges in the previous chapter, this chapter gives an overview of terms that will be used in later chapters. Also, important concepts, employed from previous research, are briefly described with representative examples.

2.1 Overview

Section 2.2 describes the systems engineering approach, including the definition of model-based systems engineering, architecture and architecting considered in this research work, and the details of the in-house architecting tool developed in the research group of which the author is a part. Then, the concept of a computational workflow is presented with the descriptions of its elements in Section 2.3, which also provides the details on how numerical solvers are employed to solve the workflow.

*The term - 'support' refers to methods or enablers developed in this research.

2.2 Systems Engineering

A comprehensive structured approach is needed to tackle today's complex technical challenges of designing and developing highly sophisticated systems such as satellites, cars, ships, aircraft, etc. The systems engineering provides a systematic way to support development of the complex systems. The definition of systems engineering (SE) given by the International Council on Systems Engineering (INCOSE) is as follows: 'The interdisciplinary approach and means to enable the realization of successful systems' [15]. The SE guides the engineering of the complex systems[7]. Here, the guiding is described as leading, managing or directing and showing the way, and the system is defined as a system of interdependent components working together to achieve a common objective. INCOSE defines a system as a combination of interacting elements and viewed in relation to its function. However, Kossiakoff and Sweet[7] postulate that a system should be sufficiently complex, requiring SE to design, develop, integrate, test and evaluate the system. Here, the complex system is a system in which its components are diverse and have an intricate relationship with one another such that its properties, capabilities and behaviours are not easily discernible.

2.2.1 Model-based Systems Engineering (MBSE)

MBSE is a model-centric approach to systems engineering. It is defined as 'the formalized application of modelling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases' [15]. It enables engineers to swiftly understand design alteration impacts, communicate design intent with other engineering teams and analyse a system design before it is built. It drives a consistent specification; also, it analyses and interrogates the system design and automates the design activities.

2.2.2 Architecture and Architecting

A system can be described from different perspectives, which result in different views of the system. These, for example, include requirements, functional, logical, physical, operational etc. views of the system. Together these views constitute the system architecture. Rechtin[7] defined architecting as structuring to bring form to function, order out of chaos, or convert the partially formed ideas into a workable model. In this thesis, the process of creating these views is referred to as 'architecting'.

In this research work, the model-based design approach to systems engineering called RFLP (Requirements - Functional - Logical - Physical) is employed. Here, it is considered that the architecting process takes place in the R, F, L and P domains. The scope of the research is restricted to the R, F, and L domains of the system.

2.2.3 AirCADia Architect

AirCADia Architect is an in-house model-based architecting tool, developed by the Advanced Engineering Design Group at Cranfield University[16]. It is implemented in an object-oriented structure. AirCADia Architect employs the RFLP approach described above. Initially, the tool focused mainly on F and L domains of the architecture and the interface between them. The purpose of the tool was to enable the architect to interactively create F and L views of the system.

The architecture in a functional (F) domain contains description of decomposition of system functions (called FHV (Functional Hierarchy View)) and inter-relations between these functions (called FFV (Functional Flow View)). Similarly, a logical domain architecture describes system decomposition (called LHV (Logical Hierarchy View)) and inter-relations between the system components (called LFV (Logical Flow View)).

The presented research has extended this approach to include dependency analysis of architectural elements (see Chapter 4) and more automated assessment of the architecture (see Chapter 5). Furthermore, the object model (which describes the framework) has been extended to incorporate the architecture in R (Requirements) domain and also a

notional computational view that describes the sizing workflows of computational models. It should be noted that the presented research work was not focused on development of methods to support architecting in R view.

2.3 Computational Workflow

This section describes the computational workflow and its elements in the context of the presented research.

2.3.1 Variables and Models

A variable is an entity that holds the information describing the value of the property that the variable represents. In this research, the variables are also referred to as parameters, and only variables describing numerical data (single value or multiple values (i.e. an array)) are considered. A computational model is an executable computer program that captures characteristics and behaviour of the system under consideration. The model takes one or more inputs (variables) and produces one or more output (variables) as shown in Figure 2.1. The models can be steady state or transient. The steady state model captures the time-independent behaviour of the system under consideration; while, the transient model captures the time-dependent behaviour of the system.

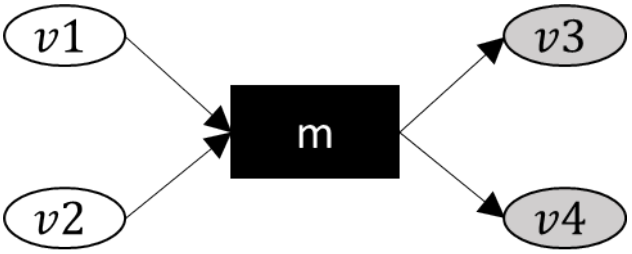


Figure 2.1: *Example of a model.*

The model is considered a 'blackbox' or a 'whitebox' model, depending on whether the model-contents are exposed. In case of 'blackbox' models, model-contents are unavailable and only inputs to and outputs from the model are available. On the other hand, the model-contents of 'whitebox' model are exposed to the user. The models may contain

information or knowledge that is regarded as closely guarded intellectual property. In such cases, the models are shared in the form of 'blackbox'. However, the unavailability of internal contents (in case of 'blackbox' model) limits the applicability of symbolic manipulation of the model, for instance, when reversing model inputs with the outputs (discussed in next section). In this thesis, the models are assumed to be steady-state and 'blackbox'.

2.3.2 Reversed or Modified Model

A model is reversed when one or more of its default inputs are swapped with one or more of the default output variables.

During the computational design process, the designer needs to specify values of some of the variables. If one of these is an output in the model's default condition, such variable is required to be set as input to the model (as its value is known). For instance, for a default model, m, in Figure 2.2 (a), the reversed model obtained by swapping the input, $v1$ with the output, $v4$, is shown in Figure 2.2 (b) as values of the variables $v2$ and $v4$ are considered available. Here, the dotted line represents the optimisation loop on top of the default model which is required to reverse the inputs with outputs in case of 'blackbox' models.

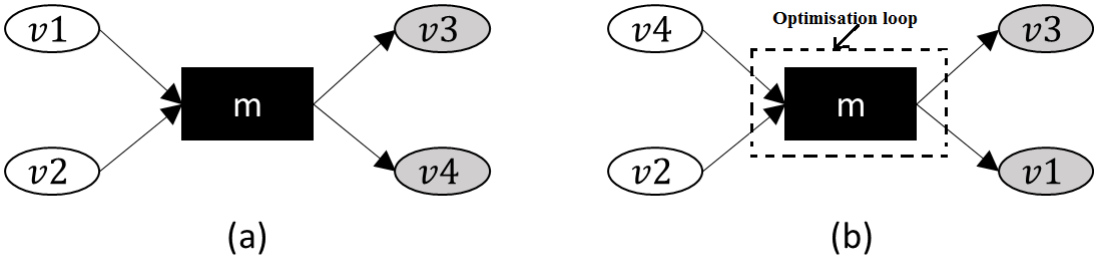


Figure 2.2: *(a) Example of a default model, and (b) Reversed model of the default model obtained swapping* $v1 \rightarrow v4$.

In the case of the 'blackbox' models, it is not possible to symbolically manipulate the internal contents (equations) of the model and change the input variable to be model's output. The process of reversing a (blackbox) model employed in this research, is described in Figure 2.3 for the models described in Figure 2.2. The process starts by

setting an initial guess value for the input that is reversed i.e. $v1$. Next, the default model, m is executed with guessed $v1$ and available value of $v2$. The execution of the model produces values of the variables, $v3$ and the reversed output variable, denoted as $v4'$ (to distinguish the value of the same variable which was available at the start, i.e., $v4$). Now, the absolute difference between the values $v4'$ and $v4$, i.e., $|v4' - v4|$ is measured. If the difference is less than the predefined tolerance value, '*tol*', then the current values of the guessed variable, $v1$ and $v3$ are supplied as outputs. If the above difference is larger than the tolerance then new values of the guess variable, $v1$, are set by the iterative numerical solver and the above steps are repeated.

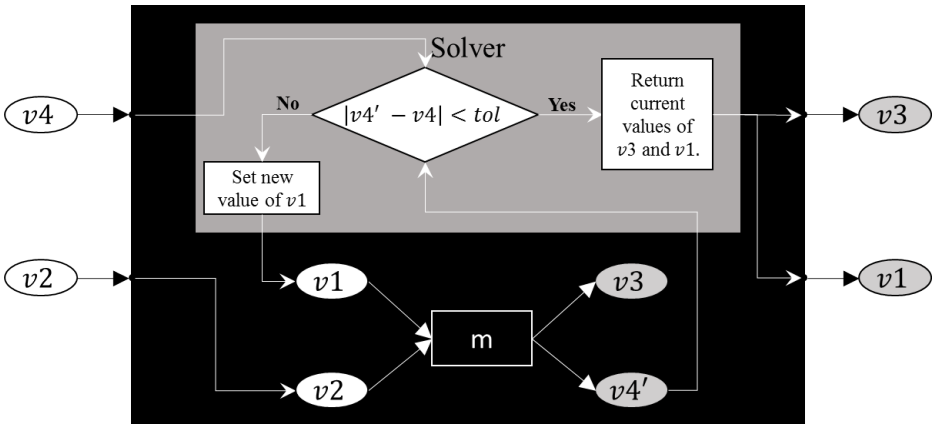


Figure 2.3: *The numerical, iterative process for the reversal of a blackbox model*[17][18].

This numerical iterative reversal process may or may not result in convergence because of several reasons, such as failure in the model execution or failure of the solver in finding the solution. These issues are not investigated in the presented research as this is not within the scope. The reader could refer to Datta's work[17] for more details.

2.3.3 Computational Workflow of Models

A workflow is a network of computational models. The models within the workflow are connected to one another through shared variables. An example of a workflow for the models shown in Figure 2.4 (a) is shown in Figure 2.4 (b).

In a workflow, the connections among the models through variables are of either feed forward or feedback type. In a feed forward type of connection, a variable calculated

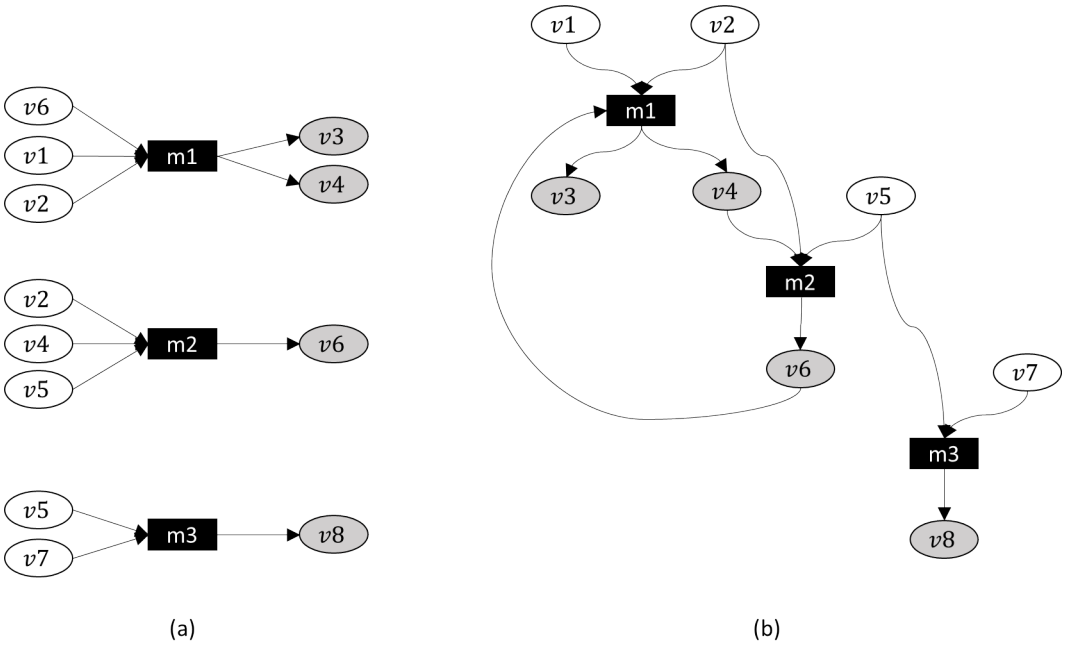


Figure 2.4: *Example of a workflow. (a) a set of models. (b) a workflow.*

by a model is input to another model in the workflow. However, in the case of feedback type of connection, a variable calculated by a model is input to another model in the workflow, where the output of this model is a direct or indirect input of the first model. The models which are connected through feedback type of connections can be described as being strongly coupled. Such a model group is referred to as a Strongly Coupled Component (SCC)[17][18]. For example, a part of the workflow containing an SCC is shown in Figure 2.5. Here, m1 has $v1$, $v2$ and $v6$ as inputs, and $v3$ and $v4$ as outputs; and m2 has $v2$, $v4$ and $v5$ as input and $v6$ as output. For the execution sequence, m1 $\rightarrow$ m2, of this SCC, it can be seen that m1 requires $v6$ as input; however, $v6$ is calculated by m2 which is executed later in the sequence. In order to solve the SCC, numerical techniques such as Fixed Point Iteration (FPI) and Gauss-Newton are employed[18]. In the presented research, methods employed by Balachandran[18] in his research are utilised.

In the FPI, at the start, an initial guess value is assigned to the unknown input, in the above example, $v6$. Then, the actual value of the variable is calculated by the model (i.e. m2) which appears later in the execution sequence, and is fed back to the earlier model (i.e. m1) as the guess value. This iterative execution is repeated until the difference

between the guessed value and the actual value obtained is below the required tolerance level.

In the Gauss-Newton method, a SCC is solved as a non-linear least square problem. In this method, the execution sequence for the models is similar as that of the FPI. However, the guess value of the unknown variable is calculated in each iteration by Equation 2.1.

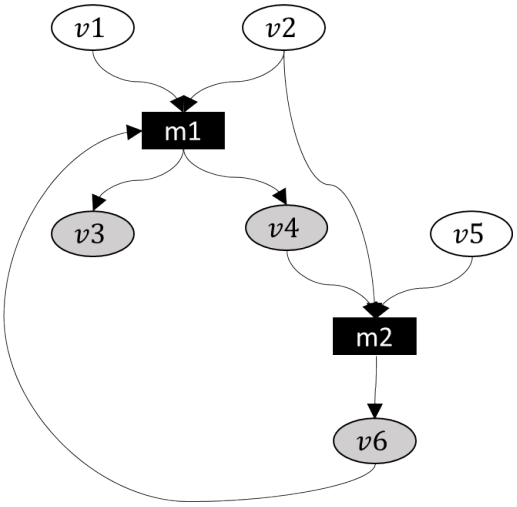


Figure 2.5: *Example of SCC.*

$$X^{k+1} = X^k - (J_f(X^k)^T \cdot J_f(X^k))^{-1} \cdot J_f(X^k)^T \cdot f(X^k) \tag{2.1}$$

where,

X - vector of feedback variables

k - iteration number

$J_f(X)$ - Jacobian of function f at X

Function, $f(X)$ contains one single sequential execution of models in the SCC with X as input and gives out a vector containing the difference of the feedback variables in the previous and the current iteration. The target is to make this output vector to zero to attain the convergence.

2.3.3.1 Under-determined, Over-determined and Determined System of Models

Depending on the number of specified independent inputs, a workflow (a system of models) can be under- or over-determined. In case of under-determined system of models, the number of independent inputs is lower than the required number to calculate all unknown variables in the workflow. The number of independent inputs is greater than the required in the case of over-determined workflow. Otherwise, system of models is determined.

In the case of system of algebraic equations, to obtain a unique solution, the number of unknowns and equations should be the same. Balachandran[18] employed similar approach to determine solvability of a set of non-linear models. However, the models can have multiple outputs unlike equations which have only one output. Balachandran[18] devised the criteria to determine the solvability of a system of models based on the principle that is employed to determine the solvability of a system of equations, as shown in the equations below.

$$TNvar - NIvar - Noutmod = 0 \quad \text{Determined} \tag{2.2}$$

$$TNvar - NIvar - Noutmod > 0 \quad \text{Under-determined} \tag{2.3}$$

$$TNvar - NIvar - Noutmod < 0 \quad \text{Over-determined} \tag{2.4}$$

Where,
$TNvar$ - Total number of variables
$NIvar$ - Number of independent variables (i.e. known variables)
$Noutmod$ - Sum of the total number of outputs of each model in a system
The term, $Noutmod$, accounts for the multiple outputs produced by the models. For example, take a system of models and workflow shown in Figure 2.4. Here, the total number of variables, $TNvar$, is 8, and total number of outputs from the models, $Noutmod$, is 4. In the workflow (Figure 2.4(b)), the number of independent inputs ($v1, v2, v5$ and $v7$), $NIvar$, is 4. It satisfies Equation 6.7 as shown below.

$$TNvar - NIvar - Noutmod = 8 - 4 - 4 = 0$$

Therefore, the workflow is determined. However, assume that in the workflow (Figure 2.4), *NIvar* is 3 which is less than the determined case i.e. 4. In such cases, additional variables are required to make the system of models determined. These are referred to as 'guessed or free variables' whose values are 'guessed' by a numerical treatment (optimiser) and the workflow is iteratively executed to satisfy some constraints on the output variables in the workflow setting up an optimisation study on top of the workflow, as described in the next section. The 'guess or free variable' is a variable which is treated as independent variable (in case of under-determined system of models) while its value is 'guessed' or estimated by the optimiser, not provided by the user.

2.3.3.2 Optimisation study

A standard form of an optimisation problem is described as below.

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0,\ i = 0, \ldots, m. \\ & h_i(x) = 0,\ i = 0, \ldots, p. \end{aligned}$$

where,
$f(x)$ is an 'objective function' to be minimised over inputs, x
$g_i(x) \leq 0$, is inequality constraints, and $m \geq 0$
$h_i(x) = 0$, is equality constraints, and $p \geq 0$

The problem would be unconstrained optimisation problem if values of m and p are zero.

Let's assume that the system of models and workflow in Figure 2.4 is under-determined with three independent inputs ($v2, v5$, and $v7$) and one 'guess' variable ($v1$). Here, values of the independent variables are known to the user; however, the value of the variable, $v1$ is to be determined, in addition to the other unknown variables of the workflow. In the optimisation study, the guess variable values (here, inputs, x) are changed within the range specified by the user. That is, the guess variable is treated as an independent

variable. The workflow is repeatedly executed for these different values (of the guess variable) along with the independent inputs untill, for instance, stated output variable in the workflow is minimised. There could be constraints on some of the variables of the workflow, as described in the form of an optimisation problem.

CHAPTER 3

Literature Review

In this chapter, a review of the relevant research areas conducted during this research is presented. The review was conducted with three considerations in mind. First, it is expected to help the author to refine the research context as well as provide the reader with a view of different aspects of the conceptual computational design. This also helps in finding avenues that would benefit from further research in the presented work. Second, it is necessary to highlight deficiencies in the state of the art and ensure that the presented research has not been done before. This is particularly useful in setting up the target research area while formulating the aim. The third purpose of the review was also to identify potential methods that could be employed with or without alteration in the current work.

At the end of the chapter, reference and impact models of the research are discussed. In addition, a success criterion employed to evaluate the success of the presented research has been carefully chosen.

3.1 Areas of Relevance

As mentioned in the introduction, this research work focuses on improving the conceptual architectural design process. In order to review relevant literature, in the current research context, relevant disciplines are identified as shown in Figure 3.1. In the figure, a discipline is shown by a hexagon. The central hexagon represents the research area. For each of the disciplines, more specific areas or topics within the discipline are shown by rectangles around the hexagon. The disciplines and topics are categorised into areas that are essential or useful for, and the areas where this research will contribute. There are areas where the presented work does not intend to contribute; however, existing methods from these areas can be useful to employ in the work to accomplish the research objectives. The literature concerning these disciplines and topics was also reviewed and is presented in this chapter.

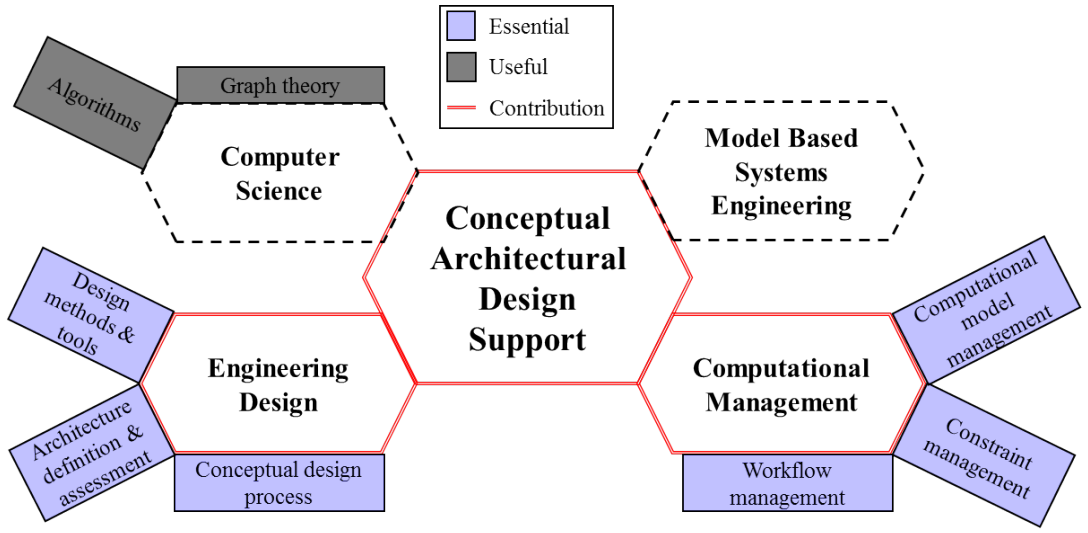


Figure 3.1: *Areas of relevance and contribution (adapted from Chakrabarti and Blessing[14]).*

The chapter is divided into three sections reviewing the literature on: architecture definition, assessment, and computational management. The chapter concludes with a summary describing the existing state of the support for conceptual architectural design, and the desired and intended condition that would improve the state of the art.

3.2 Existing Methods Supporting Architecture Definition

As described in Chapter 1 and Chapter 2, the architecture definition involves creating different views of the systems. This section reviews available approaches that support the architecture definition process during conceptual design. The review was conducted to find out state of the art in the field and scope for further research. Accordingly, the review is divided into two part: first covering the functional reasoning methods, and second covering the architecting tools and languages.

3.2.1 Functional Reasoning

As a part of the conceptual design, a functional view (F) of the system is developed. It contains decomposition of an overall system function into sub-functions (also called functional reasoning) to create its function structure. The principle structures or solutions that fulfil these functions are identified to come up with a working structure or logical view (L) of the system. Several distinct models have been developed for functional reasoning.

3.2.1.1 Freeman and Newell's Model[4]

This is the earliest of the functional reasoning models that are reviewed in this section. In this model, a structure is an entity, which has some characteristics that permit it to provide specified functions. The structure is designated with the functions it provides, and the functions that it requires in order to provide the functions, as shown in Figure 3.2. Here, F_R is a required function, and F_P is a provided function by the structure.

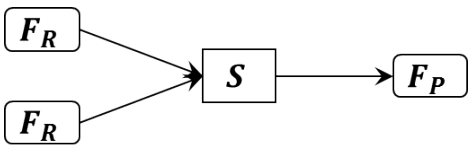


Figure 3.2: *Description of structure (S) in terms of required and provided functions.*

A set of structures and their functional specifications are taken as input, and for a given desired function, a new structure providing the desired functions is constructed combining the structures from the set. For example, for component structures shown in Figure 3.3, a new structure created as shown in Figure 3.4. Here, it can be seen that, once a function connection (connection between two structures where one provides a function, required by the other) is established, the functions involved in the connection vanish or become internal functions of the new structure (knife).

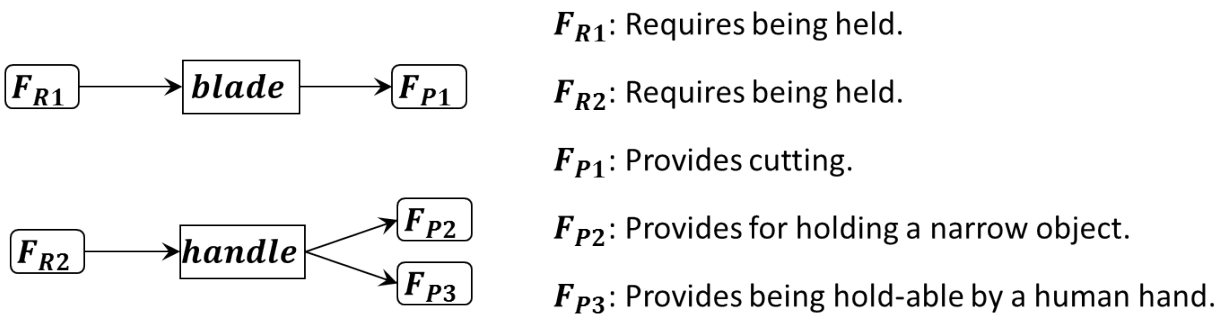


Figure 3.3: *Component structures*[4].

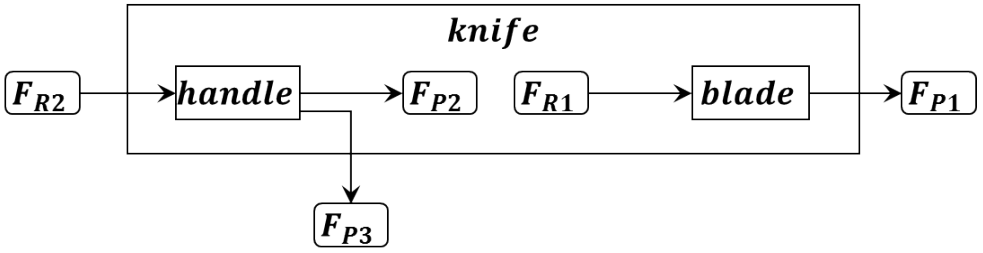


Figure 3.4: *A newly constructed structure*[4].

Similar conventions for describing structures are employed by Chakrabarti and Bligh[5] in their functional reasoning scheme (see Section 3.2.1.6).

3.2.1.2 Paradigm Model[19]

Similar to Freeman and Newell's model, the paradigm model[19] also needs a set of solutions (or structures) as input. However, the solutions are described in the form of a set of attributes which permits specific functions to be met. Here, a design problem is specified by a set of functional requirements. From the solutions set, a solution is found which fulfils the requirements best. Then, the chosen solution is evaluated to

find the components of the solution which make it not acceptable for the given requirements. Next, new specifications are generated taking the difference between the original problem and the (specifications) functions provided by the selected solution. Again, a solution from the set of solutions is searched that satisfies the new set of specifications best. The process is repeated until a combined solution is found that fulfils all the specifications of the problem under consideration.

3.2.1.3 Systematic Model[1]

In contrast to above two models, in the systematic model by Pahl and Beitz[1], functions are defined in a solution neutral way, i.e., functions are abstract formulations of tasks, described by their inputs and outputs which could be material, energy and signal as shown in Figure 3.5.

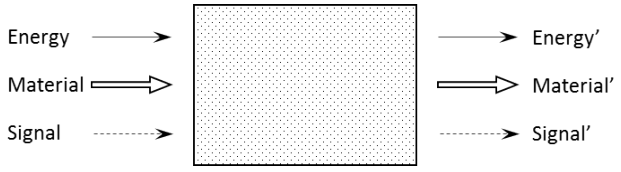


Figure 3.5: *The conversion of energy, material and signals[1]. A solution neutral function (or task) is described on the basis of inputs and outputs.*

In the model, an overall function described in the above form is decomposed further into identifiable sub-functions of lower complexity. Here, function's complexity is a measure of transparency between inputs and outputs of the function, nature of required physical processes, and the number of components/assemblies involved. For instance, a function is of higher complexity when transparency between its inputs and outputs is relatively poor, the required physical process are relatively intricate, and the number of involved components/assemblies is relatively large. The possibility of combining the sub-functions in various ways results into variants. The meaningful and compatible variants of the overall function produce so-called function-structures. A generic example of an overall function and its structure is shown in Figure 3.6, and notations used to describe the function structure are shown in Figure 3.7. Next, for the optimal function-structure, solution options to sub-functions are initiated to form solution concepts.

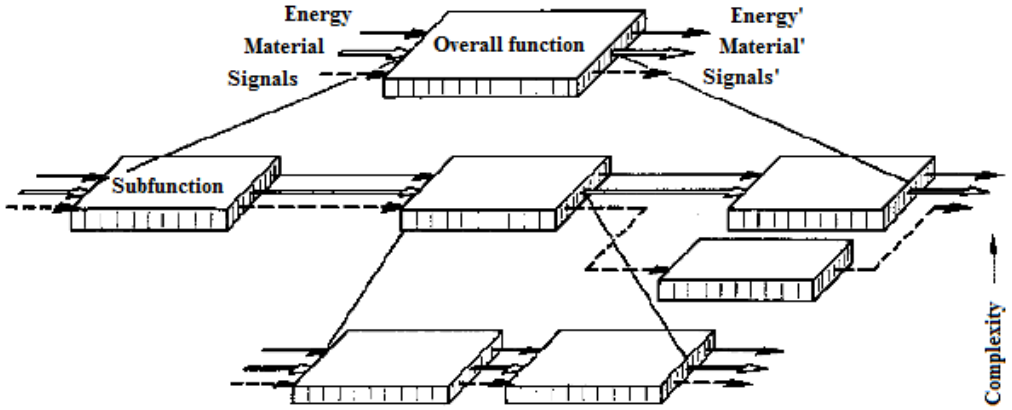


Figure 3.6: *A function structure of an overall function*[1].

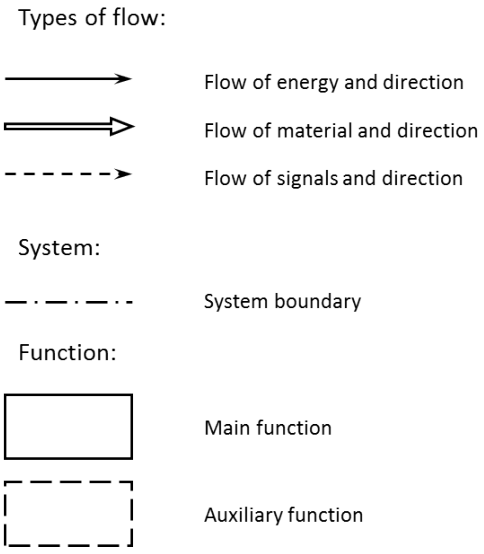


Figure 3.7: *Notations employed to describe a function structure*[1].

3.2.1.4 Function-Behaviour-State (FBS)[20]

In this model, a function is represented by an association of two ideas. The first one describes the designer's intention in the form of a verb-object pair, and other one describes the behaviour that instantiates (realises or fulfils) the function. The knowledge of existing functional design of products is stored in the form of function prototypes which is a function decomposition template. The prototype stores the information of functions, a network of sub-functions into which the function is decomposed and physical features (structures/solutions) that fulfil these sub-functions for a given existing product. These prototypes are employed to develop a functional model for a new product. This model is implemented in a software tool which is called *FBS modeller*[20].

In the tool, decomposition takes place in two phases: 1) decomposing the overall function into sub-functions, utilising the knowledge of function prototypes; and 2) 'causal decomposition' which happens after a candidate physical feature has been chosen to fulfil a particular function, where for the chosen feature there is a need of an additional phenomenon (or function) for that feature to perform the function which it is fulfilling.

Based on this reasoning model, Hamraz and Clarkson developed a method for supporting engineering change management[21][22]. However, this method has some limitations from usability point of view. From the evaluation results, it is found that building the change propagation model using this method involves a large amount of inputs and efforts. Moreover, considering the current research context, the method is limited to change propagation in functional and logical views of the architecture.

3.2.1.5 Functional Basis[23][24]

Stone and Wood[23][24] proposed a common and consistent functional design language called a functional basis to model the functionality of the products and processes. Here, functions are described in a verb-object form. The functions, similar to Pahl and Beitz's approach[1] of describing the function, are operations on flows of materials, energies and signals. In this approach, the overall functional requirement is expressed by a black-boxed operation on the flows. Then, this overall function is decomposed into sub-functions which are defined in the functional basis libraries. The library contains eight main categories of sub-functions as follows: branch, channel, connect, control magnitude, convert, provision, signal and support.

The Concept Generator, an automated, mathematically based design software tool employing this method, is used to generate functional models. It retrieves archived knowledge (information of design solutions and sub-functions fulfilled by them) to build a functional model of the product (for a given overall function).

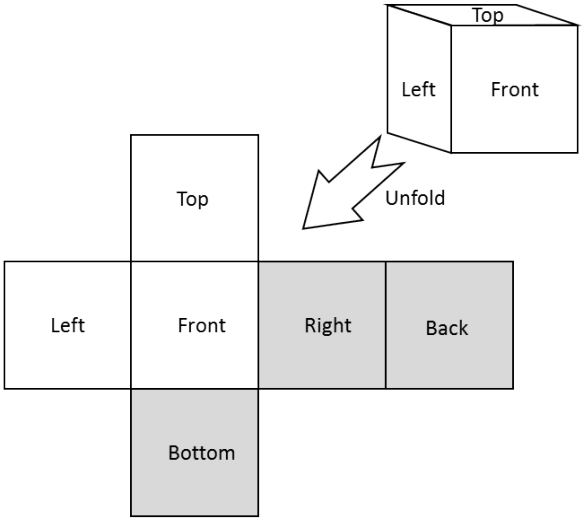
3.2.1.6 A Scheme for Functional Reasoning[5]

Chakrabarti and Bligh[5] consider that any functional reasoning model ideally should support 1) design of any nature (i.e. routine or innovative); 2) synthesise a solution obtained for a design problem at a given level of detail; and 3) expansion of a solution through subsequent levels of detail. The authors reviewed three models: Freeman and Newell's model[4], the paradigm model[19], and Pahl and Beitz's model[1]. They suggested modifications in the above models and accordingly, proposed a scheme for functional reasoning. In their model, solutions are described in a way similar to Freedman and Newell's model[4]. A set of known structures, along with its required and provided functions, are assumed available as a knowledge base. In addition, the design problem under consideration is described in terms of a set of required functions. The model contains a recursive problem definition where the initial problem is the set of required functions. The problem definition is recursively modified at each step when solutions are synthesised for fulfilling the parts of the function definition. The revised problem definition is a new set of functions that contains functions that are not fulfilled from the previous problem definition and includes newly added functions which are required functions for a chosen solution in the previous step. The process is repeated until there is no function left in the problem definition to be fulfilled.

3.2.1.7 Cube Model[25]

Zou and Du[25] proposed a new functional reasoning model called 'Cube Model' which supposedly overcomes the limitations of the systematic model proposed by Pahl and Beitz[1]. The limitations of the systematic model, as described by the authors, include 1) difficultly in describing an overall function with all inputs and outputs flows (material, energy and signal) according to demands of the customer, 2) no definite target flow considered in decomposing overall function. The functional reasoning in the cube model is based on three rules, unlike the systematic model where reasoning is based on inspiration and experience of the user. The proposed reasoning rules are inspired by the Freeman and Newell's model[4] and the 'Zigzag' model by Suh[26]. During reasoning, only one flow (material (M) or energy (E) or information (I)) is considered and

three different function-structures, corresponding to each of the flow types, are generated for a given top-level function of the system. This is shown by the three visible planes of the cube in Figure 3.8, i.e., front, left and top for material, energy and information function-structures, respectively. The invisible planes, opposite to the above visible planes, describe the concept solutions corresponding to the function-structures in the visible planes. The benefit of this model is that the reasoning is based on three rules, and therefore, can be automated using computer support.



Front: *Function-structure of M_subsystem;* **Back:** *Concept solutions of M_subsystem;*
Left: *Function-structure of E_subsystem;* **Right:** *Concept solutions of E_subsystem;*
Top: *Function-structure of I_subsystem;* **Botttom**: *Concept solutions of I_subsystem;*

Figure 3.8: *Functional reasoning cube Model*[25].

3.2.1.8 Function Analysis System Technique (FAST)[27][28][29]

FAST is a function decomposition method extended by Bytheway[30], from the functional analysis developed by Miles[28]. Here, higher level functions are decomposed into secondary or lower level functions which are represented in a form of a logic diagram, called a FAST model, using the elements described in the Figure 3.9.

While creating the model, four main questions 'how do you achieve the function?', 'why do you do the function?', and 'when you do this function?', 'what other functions must you do?'[27] are addressed. The function is expanded in two main directions i.e. 'how' and 'why' as shown in Figure 3.9. The model is built along 'how' direction by

asking 'how is the function achieved?' and the answer to this question is placed to the right of the function in the form of 'verb-noun' pair. Similarly, a question, 'why is the function undertaken?' is asked to expand the FAST model on the left of the function. Model building in the 'why' and 'how' directions happens on, as described in the Figure 3.9, 'critical path'. The model in 'when' direction describes the functions that happen with the function or the functions that are result of the function or vice-versa.

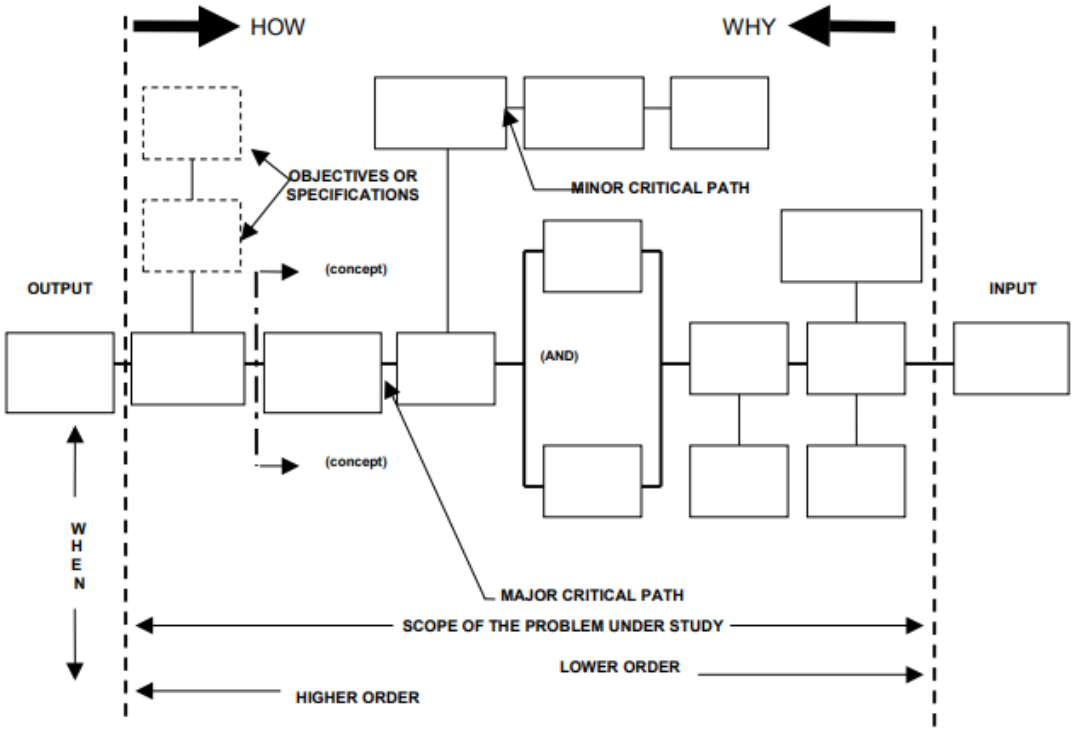


Figure 3.9: *Basic FAST model*[28].

An example FAST model for a mouse trap is shown in Figure 3.10.

FAST diagrams are usually prepared in a workshop setting. The workshop is led by the expert with experience in preparing FAST diagrams. These diagrams help the teams to reach consensus on their understanding of the system. However, FAST mainly focuses on development of a functional view of the system.

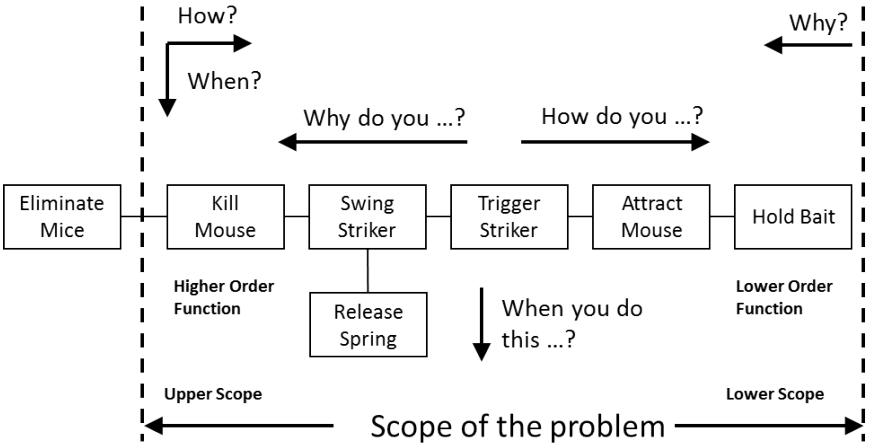


Figure 3.10: *FAST model example*[27].

3.2.1.9 Axiomatic Design (AD)[26]

Axiomatic design (AD)[26] provides a systematic process for the design of engineering systems. In AD, the design process takes place in four domains: customer, functional, physical and process as shown in Figure 3.11. This design methodology is based on two main axioms: 1) Independence axiom advises to maintain the independence of the functional requirements (FRs), 2) Information axiom advises to minimise information content of the design. The AD proposes a 'zigzagging' decomposition process. Here, for a given Functional Requirement (FR), a solution is chosen in the physical domain, and depending on the solution, derived functions are added in the next hierarchical level in the functional domain. This zigzagging process is repeated until there are no unfulfilled functions left in the functional domain. Tate[31], based on this design theory, proposed guidelines for functional reasoning. Here, several sources of sub-FRs, such as parent FR, parent Design Parameter (DP), parent Design Matrix (DM), parent (or higher level) (Constraints) Cs and (Customer Needs) CNs, are identified, and guidelines to determine the sub-FRs are outlined.

This methodology, however, has a limitation. For example, the modern engineering systems often contain multi-functional components (fulfilling more than one function at a time) which violates the functional independence i.e. independence axiom. It is not always possible to satisfy the independence axiom of the methodology while designing complex systems, such as aircraft.

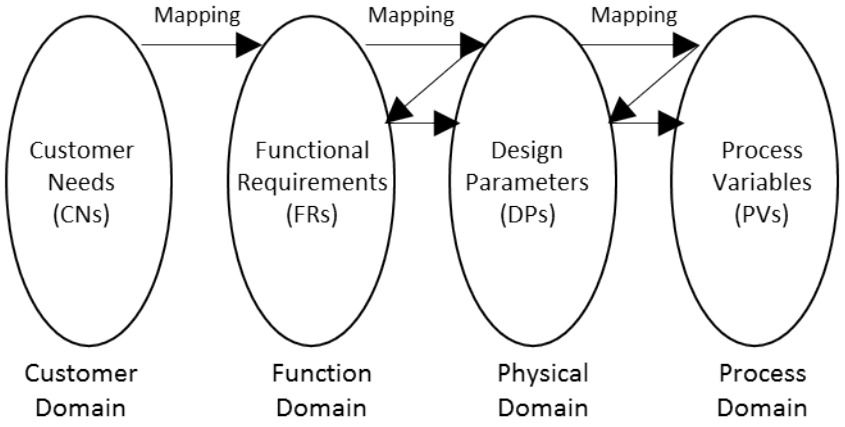


Figure 3.11: *Axiomatic design domains.*

3.2.1.10 Discussion

It can be seen from the descriptions of the models that there is not a consensus view on definition and representation of a function, and a single policy employed for decomposing a function[32]. Some models[1][23][24] contain a description of a function in a solution neutral way and in the form of a verb-object pair, whereas others[4][5][20] describe a function with reference to the solution under consideration. Due to lack of consensus, functional reasoning knowledge generated in these models differs from each other in how it is represented. This hinders the development of engineering computational tools and knowledge bases that can be employed in functional reasoning.

Despite the above differences, there are some common reasoning elements in these models, although, described by different names. For instance, required function in case of Freeman and Newell's[4], and Chakrabarti and Bligh's[5] models becomes a part of decomposed functions in next level; however, this is referred to as causal decomposition in FBS model or 'zigzagging' in the AD.

The functional reasoning models only focus on functional and logical domains of the system. While generating the functional (F) and logical (L) views of the new architecture, the user needs to navigate between the views and trace the dependency of the architectural elements which is not supported in these reasoning models. Furthermore, the assessment of the system architecture and the relations of the elements in the above (F and L) views with other views of the system, such as requirements (R) and physical

(P), are out of the scope of these models. These models also lacks dependency analysis of architectural elements.

The presented research did not focus on development of a new functional reasoning scheme. The research emphasised development of an architecting framework that would support the aspects of these reasoning models, a means for dependency analysis between the architectural elements, and methods for assessment of the architecture.

3.2.2 Architecting Languages/Tools

Several different languages and tools have been developed to support the system architecting process. A review of these is presented below.

3.2.2.1 SysML [33] [34]

SysML is an object-oriented systems modelling language developed for systems engineering applications [33]. This is an extension of a subset of the Unified Modelling Language (UML) developed by the Object Management Group (OMG) [35]. UML [36] was initially developed for the field of software engineering to describe software systems. In SysML [37], two additional diagrams, called requirement and parametric diagram, are added. In total, the SysML contains nine diagrams: activity, sequence, state machine, use case, block definition, internal block, package, parametric, and requirement, as shown in Figure 3.12. These diagrams are employed to describe the static and dynamic views of the system architecture.

SysML provides flexibility for the user in creating these diagrams, covering a wide range of applications, but on the other hand, it entails complexity. Also, the dependency between the elements of these diagrams is required to be manually traced, and thus, the approach lacks interactivity during the architecture definition process. Furthermore, SysML is not an executable model, although it allows the user to describe the computational models behind the structures/components of the system through the parametric diagram. The SysML4Modelica transformation specification [38] has been

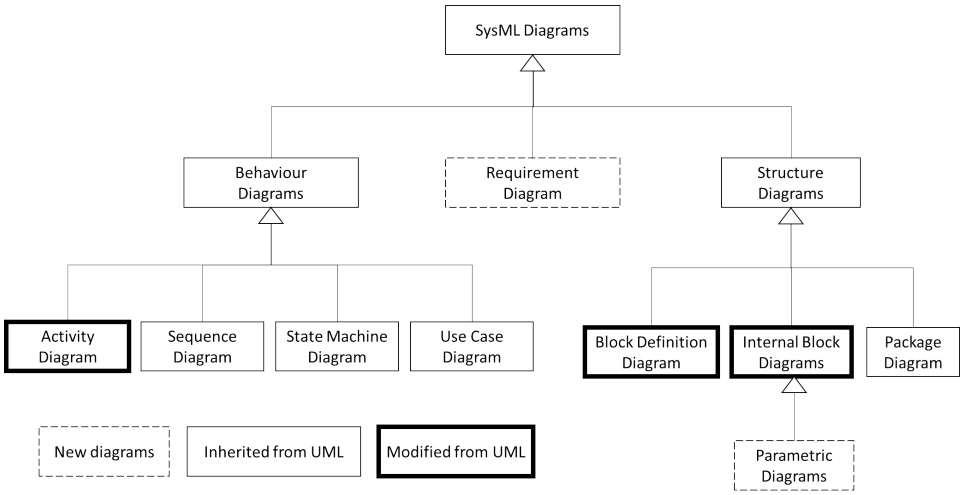


Figure 3.12: *SysML diagrams.*

developed to automatically convert the SysML model information into an executable Modelica[39] [40] model to simulate the system behaviour. However, as SysML does not compile the parametric diagram, it is possible to end up with an erroneous Modelica model.

3.2.2.2 Architecture Analysis and Design Language (AADL)

Architecture Analysis and Design Language (AADL) was developed by SAE International sponsored committee of experts and was approved and published in November 2004, as a SAE Standard AS-5506[41]. It is an architecture language and focuses on design specification using a rich, formal semantics, which are used to analyse and generate the system[42]. Figure 3.13 shows the focus of both AADL and SysML[37] in capturing the architecture of an embedded system[43]. AADL is primarily designed to capture the hardware and software aspects of a system and interactions among them.

As pointed out by Delange[42], although AADL provides precise semantics, interchange format and multi-domain analysis code generation support, compared to SysML, it has a steeper learning curve. It appears that the focus of the AADL is on real-time embedded systems where sensors and actuators are tightly coupled with software components.

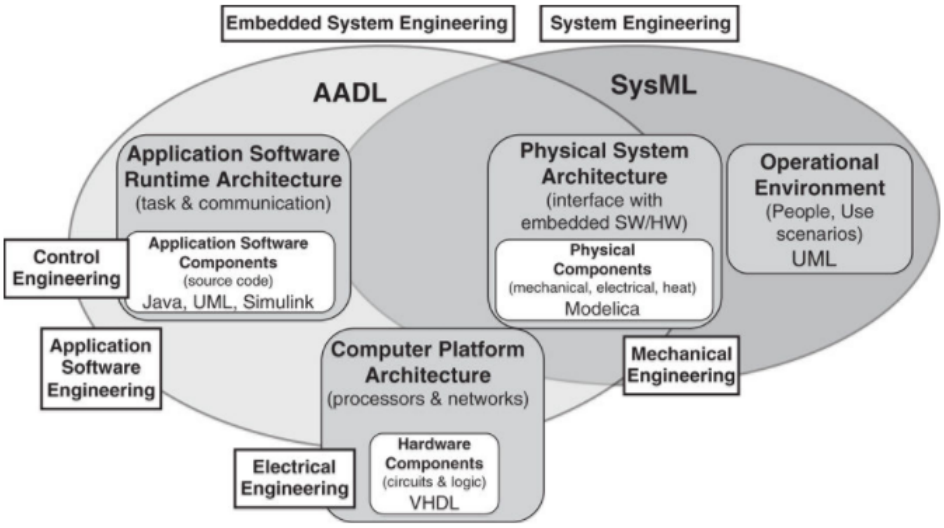


Figure 3.13: *AADL vs SysML*[43].

3.2.2.3 AirCADia Architect[44]

AirCADia Architect[44] is the architecting software developed by the Advanced Engineering Design Group at Cranfield University[16]. It is assumed that the architecting process takes place in four domains: Requirement, Functional, Logical and Physical (RFLP)[45]. Currently, the tool is only focused on interfacing functional and logical views of the architecture. The functionalities provided by the tool allow the user to add/delete a function/component of the architecture, and establish mapping/fulfilment relations between the elements of the F (Functional) and L (Logical) views. This tool does not support traceability of the modifications on other architectural elements.

In this research, this tool is employed to implement the proposed methods. For this purpose, the object model of the tool is modified to incorporate the new methods. The modifications involve addition of new classes and changes to existing classes; however, some classes are used without modifications (for more details see Section 6.2.1).

3.3 Complex System Sizing

3.3.1 Aircraft Sizing Methods

There are several methods for aircraft architecture sizing and finding the aircraft performance. The review of the relevant methods within the context of the presented research is described in this section.

One of the earlier tools called Flight Optimisation System (FLOPS) was developed by NASA[46] to design and evaluate advanced aircraft concepts. FLOPS was specifically devised for the conceptual design stages. It is comprised of six modules focusing on, 1) weights, 2) aerodynamics, 3) propulsion data scaling and interpolation, 4) mission performance, 5) take-off and landing, and 6) program control. The models employed in FLOPS use empirical equations, i.e., equations derived from data on existing aircraft. This limits the applicability of the tool to novel aircraft where the equations are no more valid. Furthermore, the sequence of execution of these models is pre-specified, requiring particular inputs. Therefore, it is not possible to re-arrange these models for different design exercise involving different inputs. Moreover, the tool does not have any link with the evolving aircraft architecture unless a human does relevant changes in FLOPS to accommodate architectural changes. Nevertheless, FLOPS has already proved its usability for designing (conventional) aircraft which are derivations of the existing aircraft.

Liscouete Hanke[47] proposed a model-based methodology which integrates aircraft sizing and analysis as shown in Figure 3.14. The method is comprised of three concepts: logic-tree, operational design space concept and power system module. The sequence of systems of the aircraft are organised for their sizing as follows: power consumers, distribution and transformation, and generation systems. These systems are modelled for both sizing and performance evaluation, using a logic-tree of the system under consideration. The logic tree describes global parameters and constraints associated with the system, system parameters, coupling with other systems and calculated parameters from the system. The trees for each of the systems of the aircraft are created using expert

knowledge obtained through interviews with designers in the field. The involved models are assembled automatically for the sizing of the systems; however, it is not clear how models within the system are orchestrated, how coupled systems are handled and how under-determined or over-determined workflows are dealt with in the approach. The employed models are developed in Modelica and therefore encounter the limitations of Modelica within this research context, as discussed in Section 3.3.3. Moreover, the setup of system sizing problems appears to be performed manually. Nevertheless, the effective integration of aircraft sizing and performance evaluation with model-based design approach is undoubtedly beneficial to explore different architectures during conceptual design.

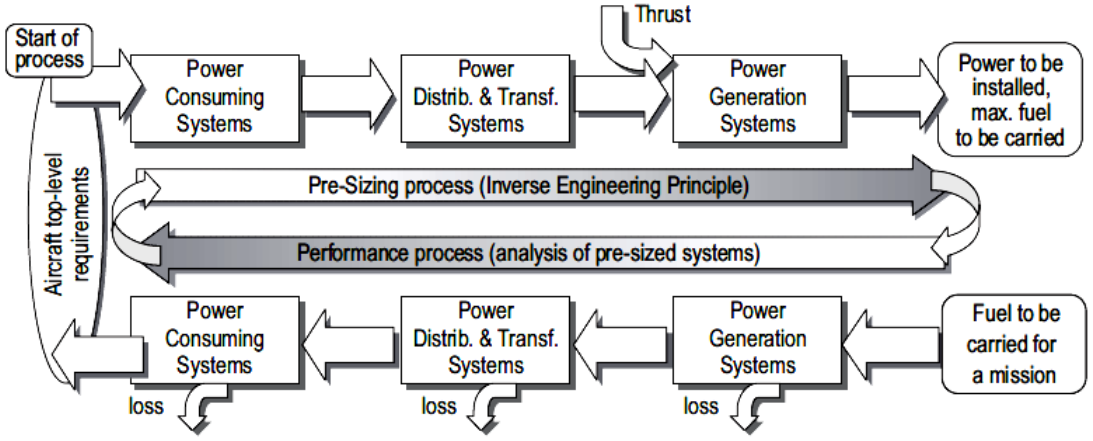


Figure 3.14: *Liscouete Hanke's framework for integrated sizing and performance process*[47].

Chakraborty et al. developed a method for aircraft conceptual design[48][49][8][11], which integrates sub-system architecture sizing with analysis. The method is capable of dealing with novel sub-system architectures. The overall integrated analysis environment is shown in Figure 3.15, where there is a block in the middle for sizing each of the sub-systems. However, it is not clear how the sub-systems are orchestrated for their sizing. Furthermore, the computational sizing workflows are manually created. Also, this method solely focuses on system analysis for a given aircraft architecture and not on the architecture definition (architecting). The architectures are created in isolation and mainly within the logical view of the system. It appears that the architectural data is manually transformed into the information required for assessment of the architecture.

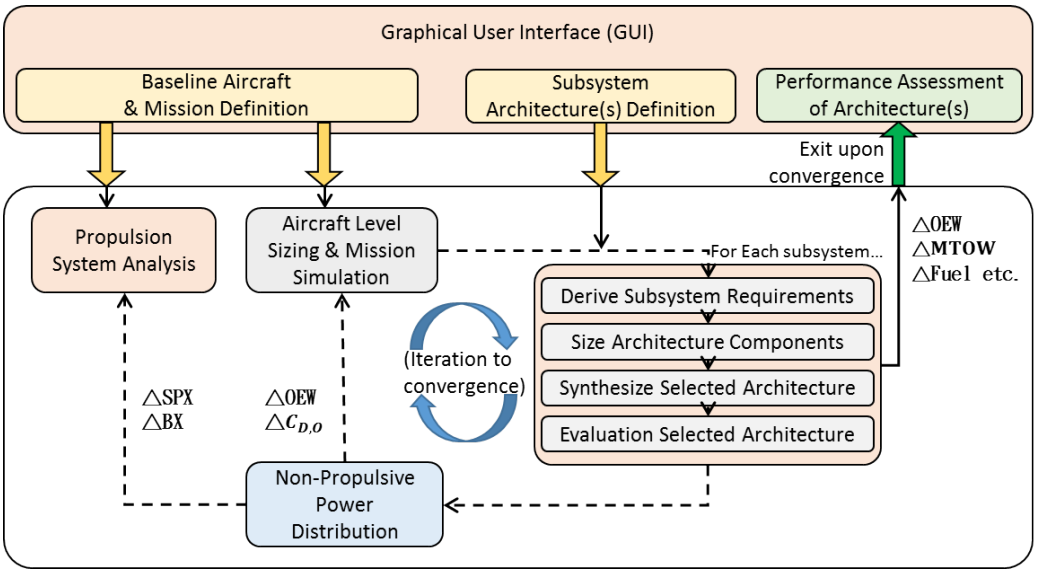


Figure 3.15: *Overview of the integrated analysis environment proposed by Chakraborty*[10]

A framework that supports automated generation of multiple concepts (in the logical view) and their analysis during conceptual design of the aircraft is proposed by Judt and Lawson[50] [51]. The framework is shown in Figure 3.16. Here, all concepts of the system are automatically generated using a morphological matrix (which stores information of functions and possible solutions fulfilling the functions). The information required for analysis of the architectures is a priori collected into a design knowledge database. This also includes compositional and connectivity information of the component models. During analysis, the connectivity between models is required to be reviewed continuously, until the adequate level of confidence is reached. Then, the rest of the analysis can be executed automatically, and therefore, involves initial manual activities in organising the models. It is not clear, if the models behind the components are always run in their default condition or depending on chosen independent input variables of the models are re-organised, and if this is performed automatically or manually. It is necessary during conceptual design to explore the architectural design space thoroughly to avoid chances of missing good designs. However, automatic (without human involvement) generation of a large number of concepts may involve undesirable architectures. Analysis of such concepts is unnecessary and involves large expenditure of resources. Moreover, it appears that the required data for analysis of the architectures is manually collected and stored in the framework. There is not an explicit link with the

architecture definition process; therefore, it is not possible to trace the dependency of an element from computational (analysis) domain to other architectural elements either.

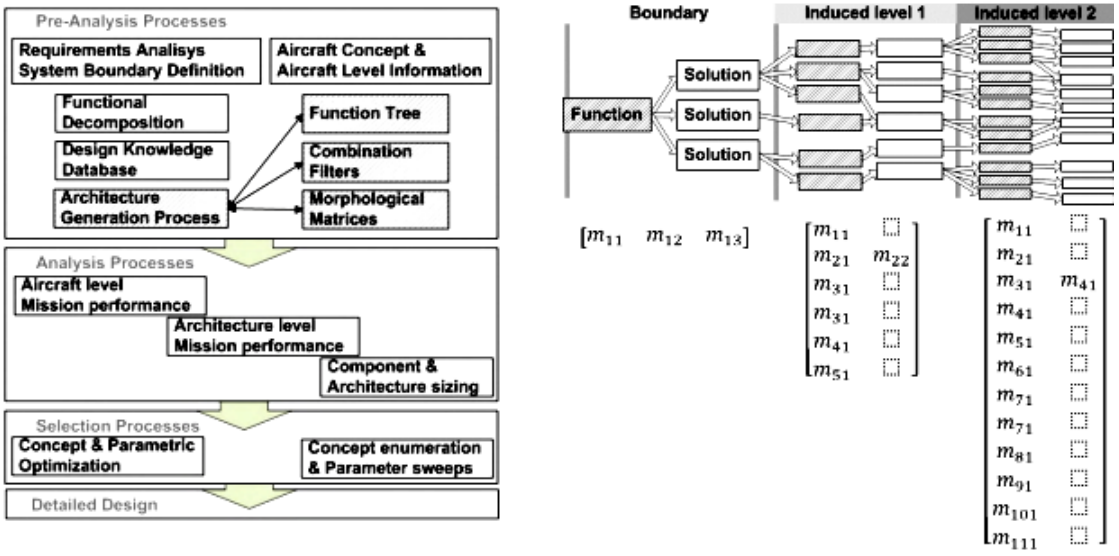


Figure 3.16: *Framework proposed by Judt and Lawson*[50][51].

A methodology for aircraft architecture sizing is proposed by Tenorio et al.[52][53] as shown in Figure 3.17. The focus of this research was to propagate changes made at component level in the architecture, to system-level. This approach covers most aspects of the conceptual system design including architecture definition[54][55], automated generation of models from architecture definition, consideration of mission and failure modes, etc. Here, the system i.e. the aircraft level is referred to as architecture level, whereas, the sub-systems level is referred to as system level. The sizing process utilises multi-level optimisation. It appears that the optimisation problems are manually formulated and set up. It is claimed that the method automatically constructs the computational model from the architecture. However, there is no explicit demonstration of the construction of workflow (a network of models) for sizing. It is not clear how this approach deals with situations involving an under-determined, overdetermined and determined system of models with multiple inputs and outputs as well as multiple models (in their default condition) producing the same output (this is possible because models may be developed by two different individuals). Moreover, SysML is employed to define the architecture, and therefore, the method retains the SysML's limitations, as discussed in Section 3.2.2.1. Within the context of current research, this method

does not focus on supporting the user during architecture definition, in particular, the traceability of architectural modifications.

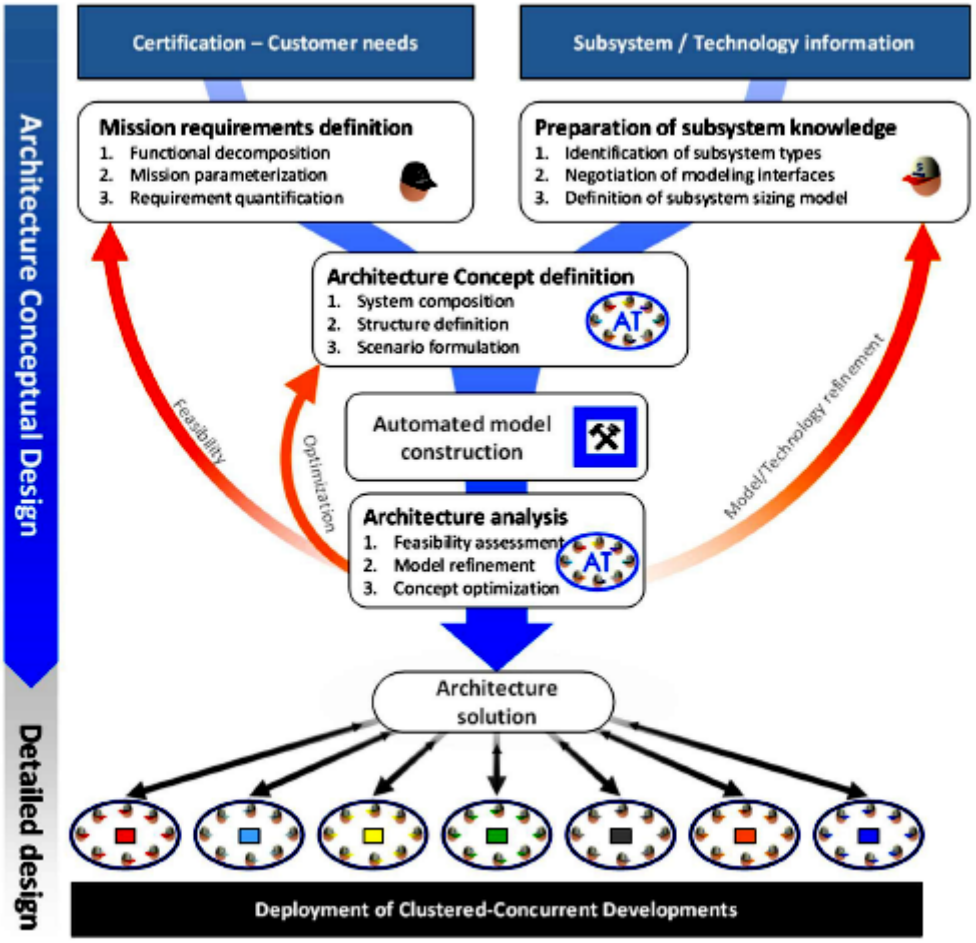


Figure 3.17: *Overview of the process*[53].

3.3.2 Other Sizing Methods

Analytical Target Cascading (ATC)[56][57] was initially developed for propagating system targets through a hierarchical system to obtain the feasible design of a complex system. It has also been used as an optimisation strategy[58]. In ATC, a system design problem is usually partitioned into a hierarchy of system, sub-systems, and components, as shown in Figure 3.18. This method needs a set of analysis or simulation models that predict the responses of the system, sub-systems and components as a function of design variables. During the ATC process, the system design targets are propagated down in the hierarchy. Then, optimal design variables of the sub-systems and components are

chosen to meet the respective targets as closely as possible and responses are given back to the system level. Depending on the responses, the targets are readjusted and the above process is repeated until consistency is obtained between the targets and responses.

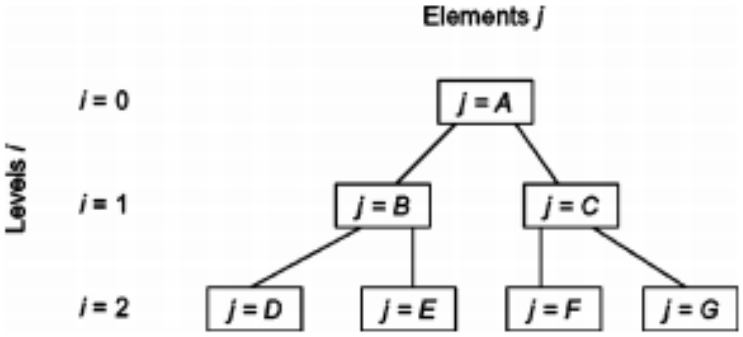


Figure 3.18: *Generic example of index notation for a hierarchically partitioned design problem of a complex system*[56].

Similar to ATC, Collaborative Optimisation (CO) is also used to design complex systems and as an optimisation strategy. However, the main difference is that, unlike ATC, the CO requires only two levels of the hierarchy of the system design problem. ATC and CO have already been applied to designing automotive[59] and aircraft systems[60].

Within the context of this research, ATC and CO can be employed to size complex systems. However, a human user has to specify the decomposition of the system design problem. Furthermore, in these methods, optimisation studies are required to be set for each of the elements of the hierarchy. These problems are set manually. Next, the organisation of the studies and the required coordination strategy is decided by the user. For every small change in the system architecture, the formalisation of the system design problem is required to be checked and modified if needed. Considering the amount of time involved in the system design problem formulation and frequent architectural modifications during conceptual design, repeating the above involved tasks manually is a tedious and time-consuming process.

Similar to the framework by Judt and Lawson[50][51] (see Figure 3.16), Helms and Shea[61][62] proposed a computational framework based on FBS model of functional reasoning to automatically generate solution concepts using object-oriented graph grammars and 'Boolean' satisfiability. Four rules are proposed and used to synthesise FBS product models. This method is demonstrated on synthesis of automotive hybrid

powertrains. This work is further extended to parametric design synthesis (sizing) of the generated concepts using constraint solving[63] [64] [65]. The approach is shown in Figure 3.19. Both the automated concept generation and parametric design synthesis are demonstrated on only one of the automotive sub-systems. Therefore, this does not validate the method's capability to deal with large systems, containing a large number of constraints and unknown parameters. Furthermore, it is not clear what kinds of models are employed in the sizing of the concept. This framework also encounters similar limitations to Judt and Lawson's method.

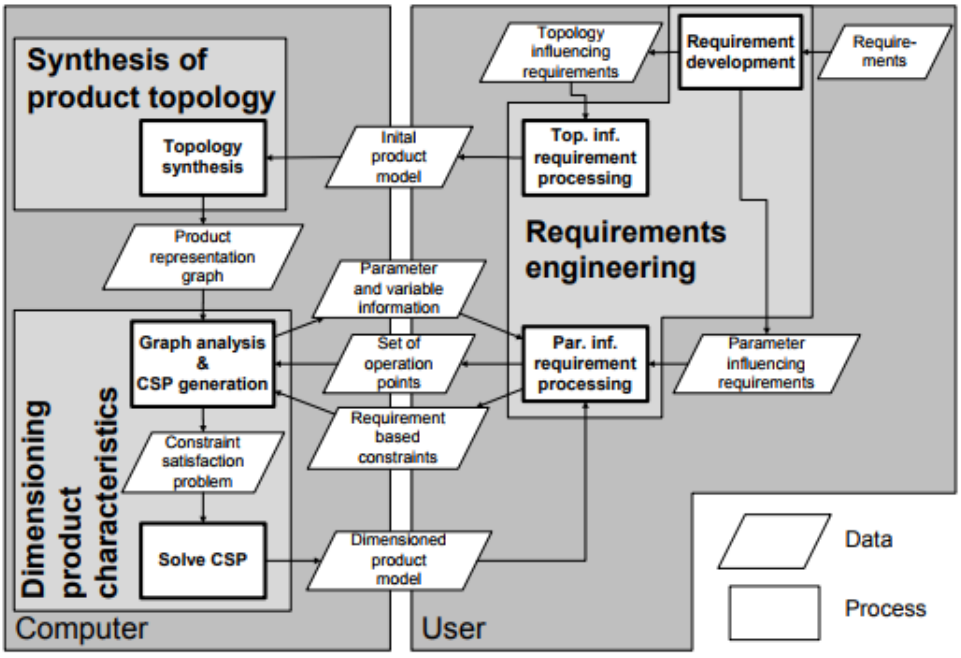


Figure 3.19: *Overview of the process*[64].

Most of the reviewed architecting tools/languages (in Section 3.2.2) only create static models of the system. That is, these models depicts collaboration and flow of data between system components in a static way and therefore, fail to describe how the system behaves in a real operational scenario. These models cannot be verified or validated because the relationship and information flow between the components are described in a static way. Wang and Dagli proposed a concept of executable system architecture to determine how the system behaves in operational environments[66] [67] [68]. The schematic of the approach is shown in Figure 3.20. Here, the system architecture described in SysML is converted into an executable model in a 'Colored Petri-Net' (CPN). The elements of CPN are mapped to the sequence diagram of the system architecture in

SysML. The CPN model simulates the behaviour of the system. However, the method does not size the architecture.

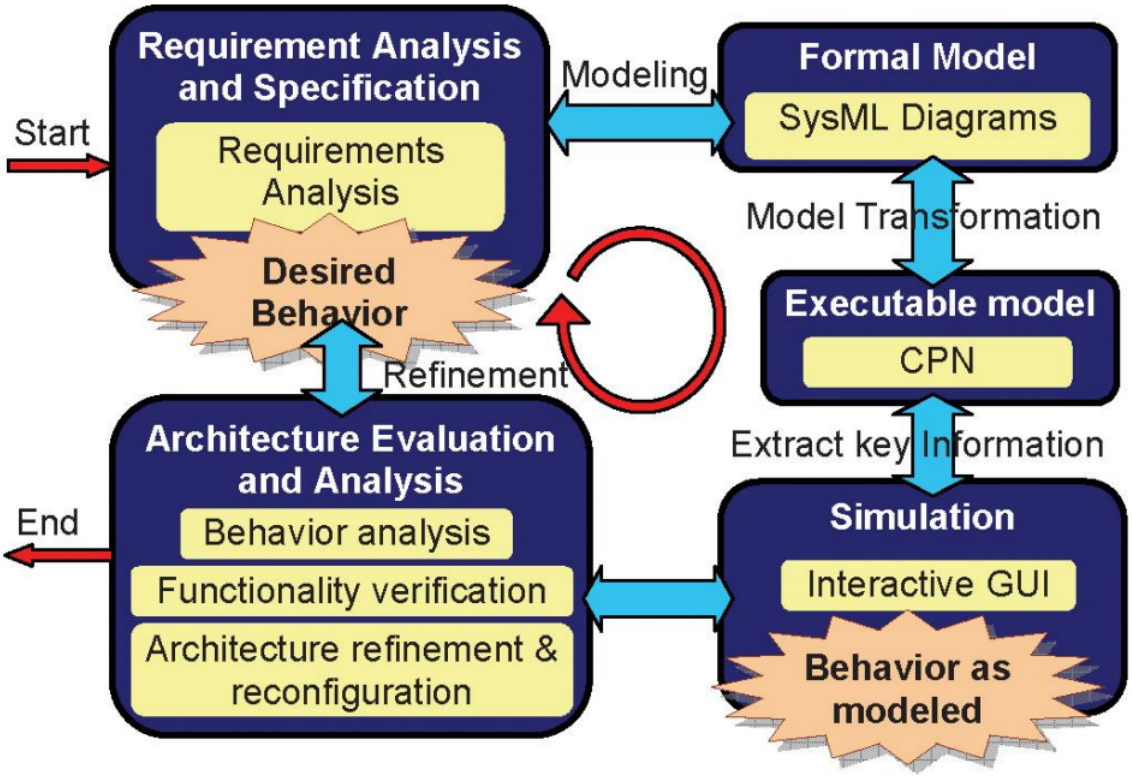


Figure 3.20: *Executable architecture paradigm*[67].

3.3.3 Software Tools/Languages

This sub-section describes the existing software tools that support sizing and analysis of system architectures.

The **Pacelab Suite**[69], a preliminary design platform, is a collection of tools, including Knowledge Designer (KD) and Engineering Work Bench (EWB). KD allows the definition of models (equations), while EWB describes the physical geometry of the system under consideration. The Pacelab Suite builds workflow of equations. The workflow describes the inter-relations among these equations and their associated parameters. It appears that this approach does not consider computational models having multiple inputs and outputs. The workflows are configured based on the parameters specified as inputs; the methods employed in doing so are propriety. The tool allows

the user to reverse inputs with outputs. However, a featured Resolution Scheme, identifying inconsistencies relating to determinacy of the syste m of equations, i.e., over- or under-determined system, only checks whether for user-selected input parameters, the system (of equations) is consistent. The tool does not guide the user in resolving inconsistencies.

Pacelab APD and SysArc, extend Pacelab Suite's design infrastructure to aircraft design. Both use KD and EWB, and therefore, encounter similar limitations as that of the Pacelab Suite. Furthermore, in the current research context, sizing the aircraft system involves many unknown parameters and finding their value while constructing a workflow containing all the equations of the aircraft (without decomposing the system of equations into problems of manageable size) may be problematic.

Modelica[39] is an object-oriented modelling language and not a tool. Previously, it was called Dymola (Dynamic Modelling Language). Unlike procedural languages, such as C and Fortran, it is a declarative language. It was developed mainly for dynamic system simulation. It is strong in simulating dynamics because of its capability for efficient solving of hybrid differential algebraic equation (DAE) systems[70]. On the other hand, steady-state models have their own importance during system design process. For instance, during design space exploration, system steady states are preferred as 'transients blur the picture for systems that are sized based on static performance'[70]. It is problematic to create and execute steady state models in Modelica. Sielemann suggested several ways to create such models in Modelica, which are different from how the dynamic models are built (in Modelica), and this requires specialist knowledge[70].

Modelica is a widely used language to model dynamics of physical systems. These models are employed in designing the systems in the conceptual stage (which is also the context of the current research).

Modelica is critically reviewed on the following characteristics, considering the research context.

- White box model: Modelica execution engine performs symbolic manipulations of model-equations. Therefore, the model should be white-box (i.e. exposing internal details of the model). This is one of the essential obligations of the model to be an acausal model (where, model is described using a set of variables and relations among them without explicitly specifying data flow, unlike causal model where inputs and outputs are specified). Therefore, it is difficult to incorporate models (EXEs, DLLs) from other modelling tools without exposing internal details into the Modelica model. Although Modelica provides the capability to incorporate models from other tools, these are always required to be executed with default inputs and outputs, i.e., in a causal way, which weakens the acausal capability of the Modelica model.

- Simulation: Modelica is a simulation language because of the way models are created in Modelica. In the model structure, it is required to specify some of the component-variables as parameters. The variables with this attribute cannot be changed during the simulation. Therefore, such (parameter type) variables that decide the size of a component cannot be output from a model. However, the model can be used to find the value of sizing-variable (parameter) by manually setting a numerical optimiser on top of this model. In the optimisation study, the model is run changing the values of the sizing parameters (values of which are not known) within a range (of values) with objective function (for more details see Section 2.3.3.2).

- Model creation: Creating a model in Modelica requires specialist knowledge. Though Modelica offers a declarative way to put equations in the model, it is challenging for a non-specialist to create locally determined and reusable models.

3.4 Computational Management of Equations, Models and Processes

The reviewed computational workflow management methods (within the scope of the research) deal with organising the system of equations/models to calculate the unknown variables from user selected independent input variables. This section describes the state of the art in workflow management methods and software tools.

3.4.1 Workflow Management Methods

3.4.1.1 Computational Process Modeller

Balachandran[18][71] developed a method which dynamically organises a system of models (with multiple inputs and outputs) to efficiently compute the unknown variables of the workflow, given a set of independent input variables. The method is comprised of the following parts: variable flow modelling, decomposition and sequencing. Various techniques are developed for each of the above parts. These contain incident matrix method (IMM) for variable flow modelling, and design structure matrix-based techniques for decomposing and sequencing of a system of models. The overall process is shown in Figure 3.21.

A design framework, based on the conceptual Computational Process Modeller, was proposed by Balachandran[18][71], to create design studies by building workflows of computational models. Given a default determined workflow of models as input, the user is able to construct new workflows by swapping variables of independent inputs with outputs of the default workflow. The default workflow is described in the form of incidence matrix, called the foundation incidence matrix. Then, a new incidence matrix for the new list of inputs after swapping of variables is produced. This is done automatically behind the scenes, involving sequential and repetitive execution of six rules[18][71]. Next, a model-to-model DSM is extracted from the incidence matrix depending on their

dependency on other models for the inputs. Then, the DSM is sequenced to determine the execution sequence of these models.

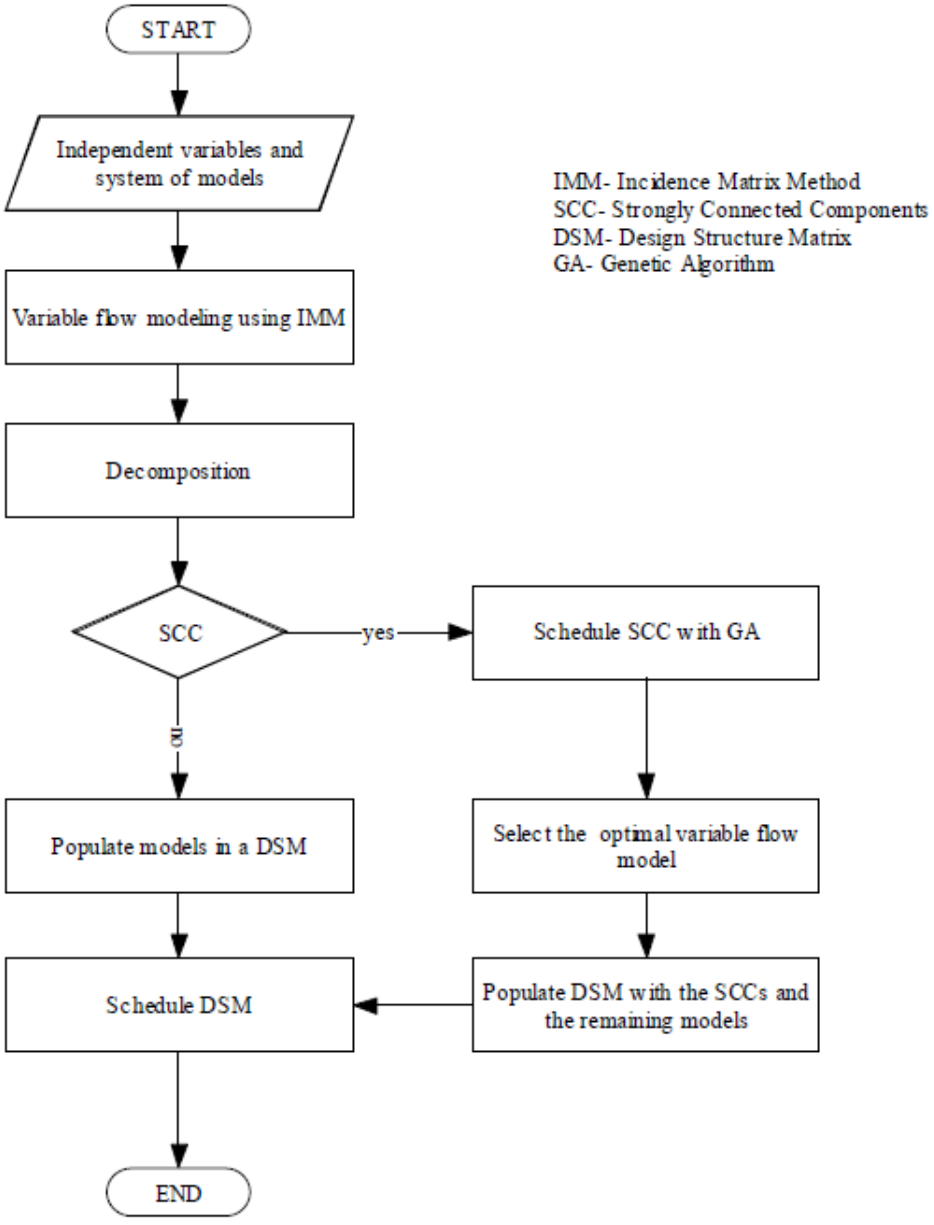


Figure 3.21: *Computational Process Modeller*[18].

The process modeller was developed with two main assumptions in mind: 1) a variable is an output from only one model, and 2) a default workflow of a system of models is available. However, during the design process, models are created by different individuals, and therefore, it is possible to have two independent models producing the same variable as output. Furthermore, a determined workflow containing such models may

not be possible during the design of the system. The process modeller was further improved by Datta[17] by introducing a method to check if a combination of variables to be swapped is correct to obtain the workflow. This method does not advise or support the user during the situations where the system of models is under- or over-determined.

3.4.1.2 Constraint Management

In constraint management, a set of equations are represented as constraints between the variables, and a network of these equations is re-configured depending on the user-specified independent input variables.

One of the earliest approaches, called 'constraint theory', was proposed by Friedman and Leondes[72][73][74] to compute, for a given computation, whether a mathematical system is 'well-posed'. Here, the constraints/equations are modelled as 'relations'. The approach allows the systematic analysis of consistency and computability of mathematical models (equations). The method automatically determines if a system of equations and its required computations are 'well-posed'. A bipartite graph (with two types of nodes - relations (constraints) and variables) was used to visualise topological insight of the constraint (equations) system, that is, equations and their relations with other equations through the variables. However, the method deals with the models with single output, i.e. equations. Furthermore, only computability of the system of equations is checked. The user is not advised or supported in dealing with the situations where the system of equations is over-determined or under-determined.

Serrano[75] proposed a graph theoretical approach for solving the system of algebraic equations. This method was applied by Rudolph and Bolling for the conceptual design of airship and FireSat[76][77]. Buckley[78] also adopted a similar approach to solving the system of algebraic equations. Similar to Friedman and Leondes[73][74], the network of equations is described as a directed graph, where, nodes are equations and their associated variables, whereas, the directed edges describe the constraint relationship of the variables with the equations. The approach did not take into consideration computational models (with multiple inputs and outputs). The workflows generated for the equations assume that a single variable is calculated by an equation, and therefore, each

equation generates one output. Although this method implicitly claims to be able to deal with constraints having multiple inputs and outputs, the outputs are represented by a single variable representing the set of all outputs, so it resembles as that of single output equations. It is not clear how a system (of models with multiple inputs and outputs), where one of the variables of the set (variable representing outputs) is associated to another constraint in the computational set, is dealt. The approach lacks detailed demonstration or testing in this regard. It does not search for different possible workflows for a given set of independent inputs.

3.4.2 Workflow Management Software Tools

iSIGHT provides a system for integrating computational models created with various software applications such as *CAD*, and *CAE*. It allows integration of models from third-party software and applications from various disciplines together in a simulation workflow. It enables automated workflow execution and exploration of the design space. Furthermore, a library of components for calling external tools, including *CATIA V5*, *Excel*, *Dymola*, *World*, *MATLAB*, *COM* and so forth, is provided. These components are the building blocks of a simulation workflow. iSIGHT also provides development environment based on *EclipseTM* which is used for developing new components. A user-friendly graphical user interface enables the user to create workflow quickly using the provided drag-and-drop mechanism. The simulation process flow created in *iSIGHT* is executed on the *SIMULIA* Execution Engine formally called *FIPER* (Federated Intelligent Product Development Environment)[79]. *iSIGHT* also offers a library of numerical treatments, such as Design of Experiments (DOEs) or optimisation, to enable engineers to quickly explore the design space[80][81]. However, *iSIGHT* does not address the requirement of conceptual design i.e. design exploration by rapidly conducting design studies because it lacks computational enablers for automated integration of models. Although, *iSIGHT* executes the computational workflow (of models) automatically, the workflow of models is required to be manually configured.

There are other similar tools, including modeFRONTIER[82], Spineware[83], TRIGs[84], Kepler[85], Triana[85], Structware[86], COSA[86], FLOWer[86] and Woflan[86], which

are used for creating workflows of simulation models or business processes. They also provide some or all of the capabilities provided by *iSIGHT* and encounter similar limitations as those of *iSIGHT*.

3.5 Reference and Impact Model

From the review of the literature relevant to the current research context, a 'reference model' that describes the existing situation in the research area is obtained as shown in Figure 3.22. The notations used in the model are defined in Appendix B. The review identified that currently, the architectural design space exploration activities are predominantly manual and there is still scope for automation, in particular, dependency analysis of architectural elements (on the architecture definition side) and orchestration of computational models/tasks (on the architecture assessment side). This is depicted in the reference model by two factors describing the number of manual and automated activities in the architecture definition and assessment process. The links between the factors are based on the authors experience, assumptions, and evidence from the literature.

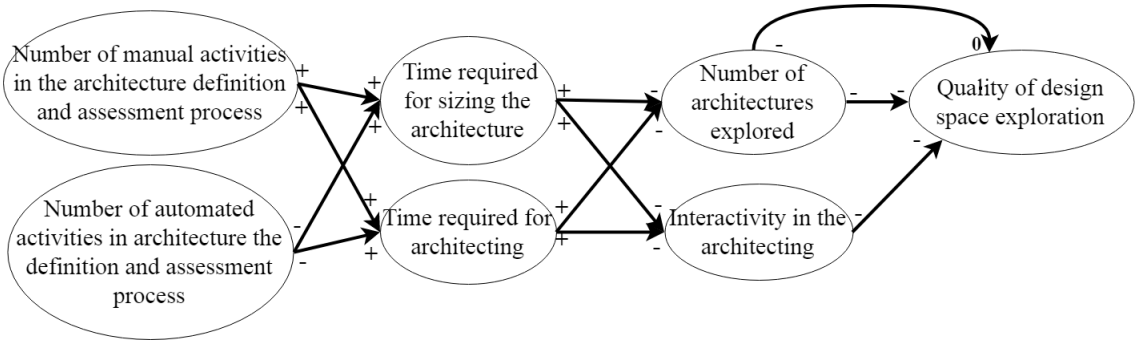


Figure 3.22: *Reference model.*

An 'impact model' that would potentially change the situation after application of the developed 'support' (or methods) is shown in Figure 3.23. The targeted factor i.e. 'key factor' is the number of manual and automated activities in the architecture definition and assessment process. Here, the methods that will be developed, attempt to reduce the number of manual activities by infusing automation in them. The 'support' of the impact model and proposed methods for it are elaborated in Chapter 4 and Chapter 5.

The success of the research is measured by its effect on 'quality of architectural design space exploration'. The quality is measured in terms of the speed and interactivity in the process. The evaluations will determine whether the developed methods support in impacting key and success factors in the intended way, i.e., increasing automation or decreasing manual activities in the definition and assessment processes.

To evaluate the success of the proposed support (methods), ideally, the support should be applied in live projects and measure the effect on 'success' factor. However, considering the limited resources, it is difficult to apply the method in the live projects and measure the 'quality of the design space exploration'. Therefore, only some of the links of the impact model are considered to evaluate the research. Evaluation of the support on 'success' factor is performed considering the links (shown in red) from 'support' to 'success factor'. However, the links between factors - 'number of architectures explored' and 'quality of design space exploration' are ambiguous because improvement in the factor, 'number of architecture explored', may or may not improve the 'quality of design space exploration', and therefore, these links are not considered for the evaluation purpose. Furthermore, only the factors (in the path of the red-coloured links) which can be measured within the project time-frame are identified, and the success of the research is evaluated by experts from industry. The experts will be asked questions related to potential effect (of application of the methods) on the 'measurable success factors' - 'interactivity' and 'number of time-consuming manual activities' in both architecture definition and assessment processes. The details of the evaluation are given in Chapter 6.

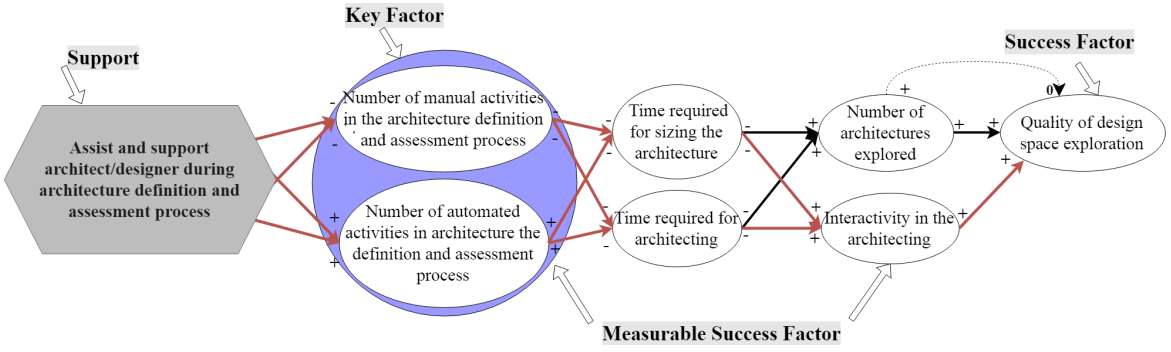


Figure 3.23: *Impact model.*

3.6 Summary and Conclusions

A review of the approaches related to the architecture definition process found that there are several architecting tools, reasoning models, and architecting languages which support definition of a system architecture. However, they do not support interactive architecture definition. This is mainly due to the complexity associated with the architecting tools and languages. These approaches do not support traceability between the architectural elements. The methods which support propagation of change require a large number of inputs from the user in the form of a matrix. Furthermore, some of the methods only focus on functional and logical views of the architecture. The above mentioned drawbacks pose the need for development of a novel method that can provide support during architecture definition. Essentially, such a method should provide an interactive way to define architectures and traceability assistance during the process. It should support the existing reasoning models to be employed, and be able to capture reasoning knowledge and reuse it in new architectures.

There are some generic methods that support sizing of complex systems, but these involve large amount of manual tasks, including the formulation of design problems, setup of optimisation studies and co-ordination strategy. A large number of conceptual methods, focused on the aircraft sizing, were also reviewed. It was found that most of these do automate some aspects of sizing during conceptual design. Most of the methods merely focus on architecture sizing and lack connection to architecture definition. That is, the user has to translate architectural information (produced during the architecture definition process) into the form, the methods take as input. Some of the reviewed methods did cover both architecture definition and analysis. However, they do not support dependency analysis among the architectural elements. Some of these methods automatically generate a large number of architectures. Here, the human user is kept out of the definition process; as a result, there could be a large number of unacceptable architectures. Furthermore, the individual sizing workflows of the sub-systems appeared to be created manually and executed in their default condition. These methods do not support automated re-configuration of the workflows for a given set of independent inputs, and also do not give advice/support in situations where the system of models is over

or under-determined. Hence, there is a research need to develop a method that could potentially overcome the limitations of these methods.

Several workflow management methods and tools were also reviewed. The majority of the methods support automated creation of workflows of algebraic equations. These methods require modifications for application in the current research context, where computational models with multiple inputs and outputs are employed. The methods, which support construction of workflows, take a default determined workflow of models as input. However, this may not be the case when sizing the system. The sizing problem is usually an under-determined problem, and the existing methods do not support under- or over-determined system of models. Where the existing tools do automatically execute the workflows, these, however, are manually created.

A 'reference model' that describes the existing situation in the research context was developed to benchmark the intended improvements in the situation. The model helped to identify the areas (factors of the model) on which to prescribe the methods (support) that would improve the situation. Accordingly, an 'impact model' was proposed, and 'measurable success factors' were identified to evaluate the research.

In the presented research, the work by Guenov et al.[44] is extended to enable the user to trace the effect or architectural changes in a view on other views, during architecture definition (see Chapter 4), and more automated sizing and performance evaluation of the architecture (see Chapter 5).

CHAPTER 4

Architecture Definition

The proposed methodology is divided into two parts and presented in two consecutive chapters. As a starting point, an overview of the methodology is provided in relation to the impact model established in Chapter 3. The rest of the chapter presents the first part, an approach to support the architecture definition process during conceptual design. This approach has been developed in an attempt to address some of the requirements of practicing architects/designers.

4.1 Overview

Attached to the key factor (as depicted in Figure 4.1) is the proposed support/methodology, ultimately aiming to reduce the number of manual tasks, during architecture definition and assessment. The methodology is divided into two parts that support - 1) architecture definition, and 2) architecture assessment processes, as shown in (Figure 4.1).

This chapter presents the methods developed for the first part, and the second part will be described in Chapter 5. In the first part, enablers are developed to assist the architect/designer in the navigation of the architecture views, in particular, inter and intra domain dependency analysis. A framework is proposed that enables the user to interactively create a new architecture. User actions on the architectural elements are recorded behind the scene in data structures, which are then, employed to perform dependency analysis. In the second part (Chapter 5), methods are proposed that automate the computational design activities involved in the architecture assessment.

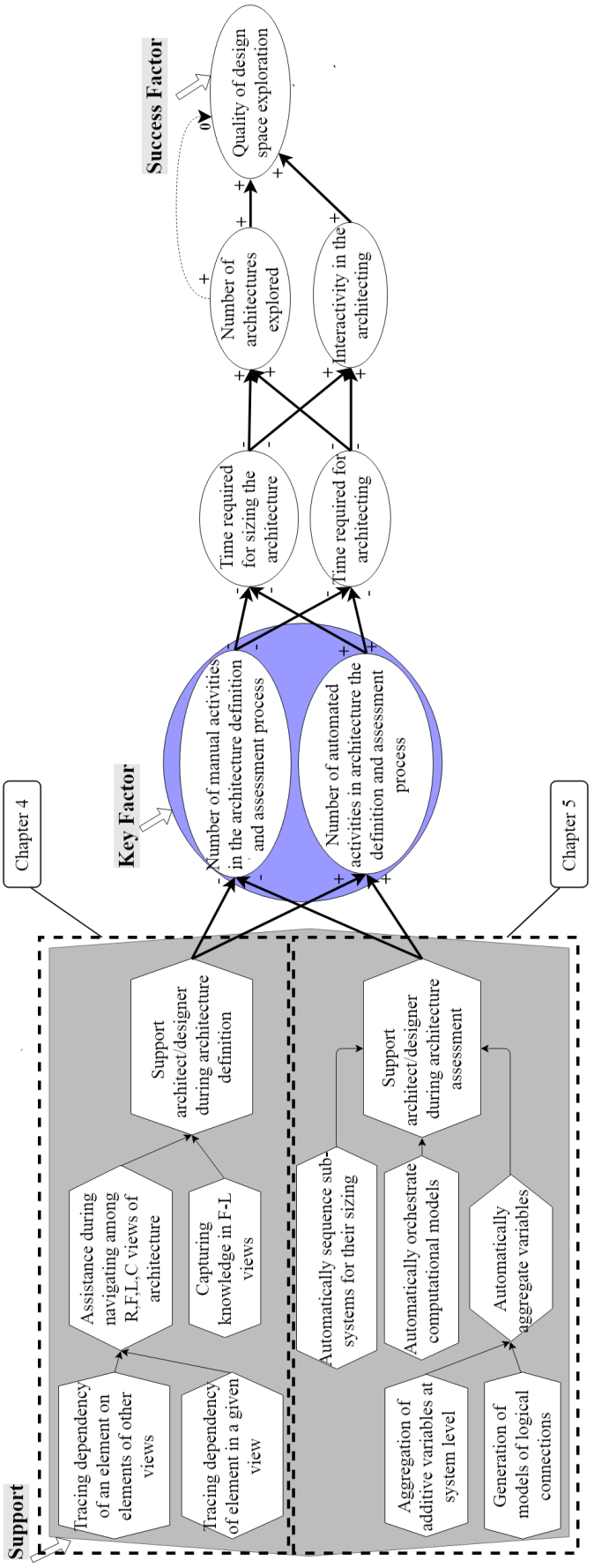


Figure 4.1: *Impact model with the support elaboration.*

4.2 Overview of the Architecture Definition Support

This chapter describes the information systems framework which supports the RFLP paradigm and in particular, the interactive operation (architecting) in the requirement, functional and logical domains. In a recent publication, Guenov et al.[44] presented data structures underpinning basic operations which take place in the F-L domains. Here, that work (in which the author was not directly involved) is extended to include traceability among architectural elements and the notion of a computational (behavioural) domain. The latter is intended to facilitate the rapid (steady state) assessment of a rapidly synthesised architecture. The details of the sizing approach are reported in Chapter 5. Here the computational domain is used specifically as a source of information needed to enable traceability of changing performance requirements. The scope of the research described herein has been restricted to the process of conceptual systems architecting, within the requirements (R), functional (F) and logical (L) domains.

The architecture definition support contains a proposal of an object model to describe the architecture, a set of data structures extracted from the object model, and algorithms that enable dependency analysis. Together, these elements (of the method) support the architecture definition process. The basic classes and their relationships are introduced through an object model in the next section. Section 4.4 formalises the relations between the architectural elements and a description of a graph-theoretic approach employed for describing the architectural views. Section 4.5 describes the algorithms proposed for dependency analysis of architectural elements. Finally, Section 4.6 summarises the chapter.

4.3 Object Model

The proposed object model consists of several classes, including: Project, Database, Architecture, Requirement, Function, Solution, Port, Model, Workflow, and Parameter. A high-level UML class diagram, describing the classes and their relationships, is shown in Figure 4.2.

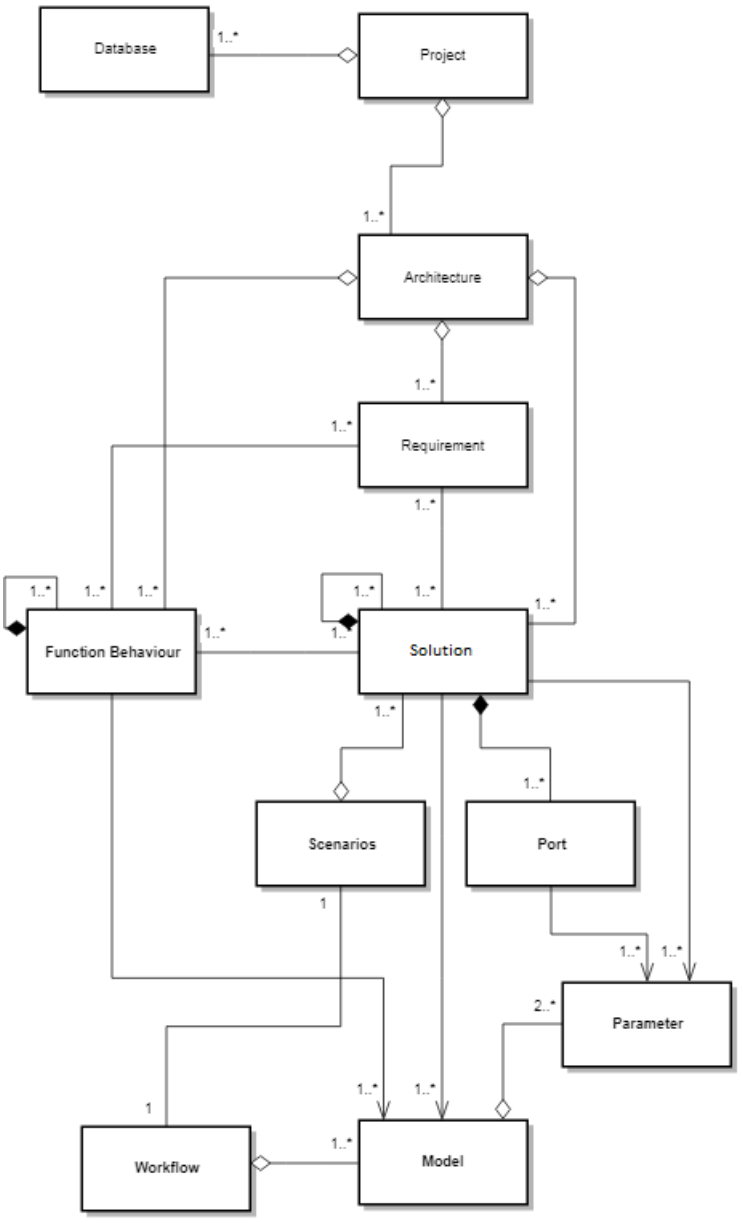


Figure 4.2: *UML class diagram of the proposed framework (attributes and methods are not shown).*

The 'Architecture' class contains the information that describes the architecture as a whole, including the list of all requirements, functions, and solutions. The 'Database' class stores knowledge about all the architectural elements (including requirements, functions, solutions, computational models, etc.) defined by the architect. The 'Project' class is a container for both the database object and the list of architectures. It is a utility class, which is used for storing studies as part of the software implementation of the framework, described in Appendix C.

The 'Requirement' class represents the technical requirements (ρ) derived from the stakeholders' needs, which could be of type functional or performance. It owns attributes and methods which enable the requirements decomposition process, and the requirement-to-function mapping relations. Similarly, the 'Function' class denotes a function, i.e., functional requirement and own attributes and methods which enable decomposition of a function. A function instance, $\varphi(n)$, is the action that a system (or product) has to perform in order to meet the functional requirement. Functions are represented in the form of a 'verb-object' pair, e.g., cool air. A function has an attribute that records a solution reference (and vice-versa) to implement function-solution mappings and derived functions.

The 'Solution' class represents physical solutions which fulfil the functions. A solution/component*, σ, is the physical object(s) that fulfil(s) the required function. A solution owns attributes that describes internal solution parameters. For instance, a compressor has compression ratio and efficiency as the solution parameters. External parameters (e.g. in case of the compressor, input/output air flow parameters - mass flow rate, pressure and temperature) are managed by port parameters, which are described below. The 'Solution' class owns attributes and methods which permit the decomposition and function-solution mapping process.

The 'Port' class describes the solution interfaces with other solutions and the environment. It describes flow type (e.g. material, energy or signal) and its direction (input or output). The Port class also owns attributes that describe information of flow parameters (e.g. in the case of a mechanical interface, 'force' and 'displacement'). These parameters offer an additional level of compatibility control between components or solutions.

The 'Parameter' class represents an engineering quantity that describes some characteristics of a solution or port in a logical domain. For instance, compression ratio, γ, is an example of a parameter associated to the compressor solution. Similarly, both (input and output) ports of the compressor may have two associated parameters, i.e. pressure, P, and air mass flow rate, $\dot{m}$.

*It should be noted that terms, 'solution' and 'component', are used interchangeably unless 'component' refers to a level in the system decomposition (hierarchy)

The 'Model' class represents the computational code (mathematical equations) for predicting the solution's behaviour or performance characteristics. Each computational model has one or more output parameters and the quantities of these output parameters are calculated using the given quantities of one or more input parameters.

The 'Workflow' class represents a network of computational models, and describes the execution sequence of the computational models. A computational model is connected to other computational model through parameters, if an output parameter of the first model is an input to the second model.

The collection of parameters, models, and workflows constitute a domain, which is referred to as computational domain in this thesis. It is important to note that the architect is not required to interact directly with the computational domain, instead he/she only needs to specify the solution computational models (which are developed by simulation specialist) for behaviours and performance characteristics. Automatic (dynamic) computational workflow creation methods[87] (proposed in Chapter 5) can be employed to maintain the computational domain behind the scenes.

4.4 Formalisation

4.4.1 Architecture Definition Process: Elementary Relations

This section formalizes the essential R-F-L mappings. The basic relations between architectural elements employed in the proposed framework are presented below. It should be noted that, if the formalisation statement is followed by a reference, the corresponding notations are taken from reference[44]; otherwise, the notations are extended by the author.

1. Mapping

   (a) Requirement-to-Function and Requirement-to-Parameter: This relation specifies the mapping from a functional requirement (ρ) to a function (φ),

i.e. $\rho_1 \multimap \varphi_1$. Similarly, there is a relation from the performance requirement to solution's parameter p_1, i.e. $\rho_1 \multimap p_1$. Here, p_1 is a solution parameter which describes the characteristics of the solution (for more details see Section 5.2).

(b) Function-to-Solution[44]: This is a mapping from functional requirement(s)(ρ) to solution(s) (σ). The following cases can occur:

- A single function, φ_1, is satisfied by a single solution, σ_1, which can be expressed as $\varphi_1 \multimap \sigma_1$.
- A single function is fulfilled by a number of (equivalent) solutions, $\varphi_1 \multimap \{\sigma_{1.1} \vee \sigma_{1.2} \vee ... \vee \sigma_{1.n}\}$, which is called redundancy.
- A set of solutions $\{\sigma_{1.1}, \sigma_{1.2}, ..., \sigma_{1.n}\}$ collectively satisfy a single function, φ_1, which can be represented as $\varphi_1 \multimap \{\sigma_{1.1} \wedge \sigma_{1.2} \wedge ... \wedge \sigma_{1.n}\}$. That is, function φ_1 will not be fulfilled if any of these solutions are missing.

(c) Solution-to-Function[44]: This is a mapping relation from a solution (or a component) to a function. Unlike the Function to Solution mapping, where only a function satisfaction relationship is possible, here two types of relationships are possible: function satisfaction and function derivation. Such relations can occur due to:

- The emergence of a derived requirement. For example, choosing a bootstrap refrigeration system ($\sigma_{1.1}$) will require the air to be pressurized ($\varphi_{1.1}$), which can be stated as $\sigma_{1.1} \twoheadrightarrow \varphi_{1.1}$.
- Assigning additional function(s) to an existing solution. That is, specifying multifunctional solutions, which can be expressed as $\sigma_i \multimap \{\varphi 1.1 \wedge \varphi_{1.2} \wedge ... \wedge \varphi_{1.n}\}$.

(d) Solution-to-model: This is a mapping relation from solution (σ_1) to computational model (μ_1), i.e. $\sigma_1 \multimap \mu_1$. Each of the solutions has models associated to it. These models mathematically describe the intended behaviours (functions) of the solution (see Chapter 5).

2. Decomposition:

- Requirement decomposition - this is the breakdown of the top-level requirements into sub-system and component level requirements, i.e. $\rho_1 \leftarrowtail \{\rho_{1.1} \wedge \rho_{1.2} \wedge ... \wedge \rho_{1.3}\}$.
- Iterative function-solution decomposition[44] - this is akin to the "zig-zagging" process. This can be represented as $\varphi_1 \multimap \{\sigma_1 \twoheadrightarrow [\varphi_{1.i} ... \multimap (..\sigma_{1.j})]...\}$. This process comprises a series of function-to-solution and solution-to-function mappings. It may also include partial single domain decompositions.
- (Invariant) functional decomposition[44] - if a function, φ_1, can be decomposed into, say three independent sub-functions,$\varphi_{1.1}, \varphi_{1.2}$ and $\varphi_{1.3}$, which are not derived functions. This can be written as $\varphi_1 \leftarrowtail \{\varphi_{1.1} \wedge \varphi_{1.2} \wedge \varphi_{1.3}\}$. Such decomposition may be representative of a 'functional flow', i.e. $\varphi_{1.1} \rightarrow \varphi_{1.2} \rightarrow \varphi_{1.3}$. Functional flow implies a clear process sequence which does not need to be directly derived from the solutions.
- Leaf function node - it is a function at the lowest level in the functional hierarchy.

3. Aggregation: Aggregation is needed when parts of the functional or logical hierarchical structures are constructed from bottom up. Aggregation happens in both functional and logical domains as described below:

   - Solution aggregation[44] is the process of combining solutions to comprise a single higher level solution (i.e. an assembly of these solutions), i.e. $\sigma_1 \rightarrowtail \{\sigma_{1.1} \wedge \sigma_{1.2} \wedge ... \wedge \sigma_{1.n}\}$. This may be the case when an integrated solution (e.g. Environmental Control System (ECS) Pack) replaces previously individually-linked solutions (e.g. compressor, heat exchanger, and turbine in the case of a ECS pack).
   - Functional aggregation[44] can be the consequence of solutions-aggregation in the logical domain.

4.4.2 Architecture Definition Process: A Graph-theoretic Approach

Presented in this section is a graph-theoretic approach to underpin the *functional reasoning* within and between the RFL domains. Graph (data) structures are constructed by using the architectural elements (i.e. requirement, functions, solutions and models) and their basic relations (presented in Section 4.4.1). These data structures serve three purposes. First, they allow changes made in one domain to be recorded and traced in the other domains. Second, they assist in defining new or variant architectures. Third, these data structures can also be used as efficient means for storing architectural information (i.e. requirement, functions, solutions, and the connections between them).

4.4.2.1 Decomposition in the Requirements, Functional and Logical Domains:

'Trees' are employed to model the decomposition (i.e. hierarchical representation) in the requirements, functional, and logical domains. A tree, T, is a directed acyclic graph which has no loops and circuits. In addition, there is exactly one root node and every child has maximum one parent, i.e. there is exactly one link between any two nodes.

A tree T is constructed by using only the decomposition relation, i.e.$e_1 \leftarrowtail \{e_{1.1} \wedge e_{1.2} \wedge ... \wedge e_{1.n}\}$, where e_i represents an element which is decomposed into n sub-elements. Each link in T represents a parent-child relationship between the two connected nodes. The elements e in T may be requirement, function, or solution, however, a tree T contains elements from a single domain only.

4.4.2.2 Functional Flow, Logical Flow, and Computational Views:

A directed graph is employed to model the functional flow, logical flow, and computational views. A directed graph (or digraph), represented by $DG(V,E)$, is a graph with a set of vertices V connected by edges E where each edge has a direction.

A directed graph DG captures the flow relations among the same kind of entities. For functional flow modelling, a graph, DG_F is proposed, which describes the functions and the flow connections between them (e.g. $\varphi_1 \rightarrow \varphi_2 \rightarrow \varphi_3$). Similarly, DG_L, describes flows between solutions and subsystems in terms of energy, material or signal. DG_F and DG_L are extracted from the information stored in the object model described in Figure 4.2. These are expressed in the form of adjacency matrices (also called as connection matrix or design structure matrix) to store the graphs and to be applied by the algorithms. The associated adjacency matrices for DG_F and DG_L are denoted by AM_F and AM_L, respectively.

4.4.2.3 Functional-Logical Zig-Zagging:

Functional reasoning employs two types of relations between functional and logical domains: 1) function-to-solution mapping, and 2) solution-to-function mapping (also called derived functions). Here, a directed bipartite graph is used to model the *functional reasoning* (functional-logical zigzagging). A bipartite graph BG is a graph with two disjoint sets of vertices V_1 and V_2 and a set of edges E where each edge connects vertices from opposite sets.

The *functional reasoning* directed bipartite graph, BG, can be expressed by Equation 4.1. Here, Φ is a set of functions,Σ is a set of solutions and E is a set of edges representing the function-to-solution and solution-to-function (derived-function) mapping.

$$BG = G(\Phi, \Sigma, E) \tag{4.1}$$

For any edge, described by its end nodes, φ_i and σ_i the following condition is true.

$$((e = (\varphi_i, \sigma_i) \vee e = (\sigma_i, \varphi i) \wedge (e \in E) \Rightarrow (\sigma_i \in \Sigma \wedge \varphi_i \in \Phi) \tag{4.2}$$

Bipartite graphs are proposed to model the *functional reasoning* for both the complete database and the individual architectures. The BG for the individual architectures (represented by BG_A) is the subset of the complete database BG_D. Figure 4.3 shows the

bipartite graph of the complete database (BG_D) on the left, and the two bipartite graphs of the individual architectures (BG_{A_1} and BG_{A_2}) on the right. It can be observed from BG_D that there are two options to fulfil function φ_1. The first option is a combination of solutions ($\sigma_1 \wedge \sigma_2$), whereas, the second option is a single solution, σ_3. However, the second option, σ_3, has two derived functions, φ_2 and φ_3. This information is useful while creating a new architecture. That is, if a solution is selected to fulfil a given function, and in turn the solution requires some functions to be available, this drives the decomposition inducing new derived (required) functions in the functional domain.

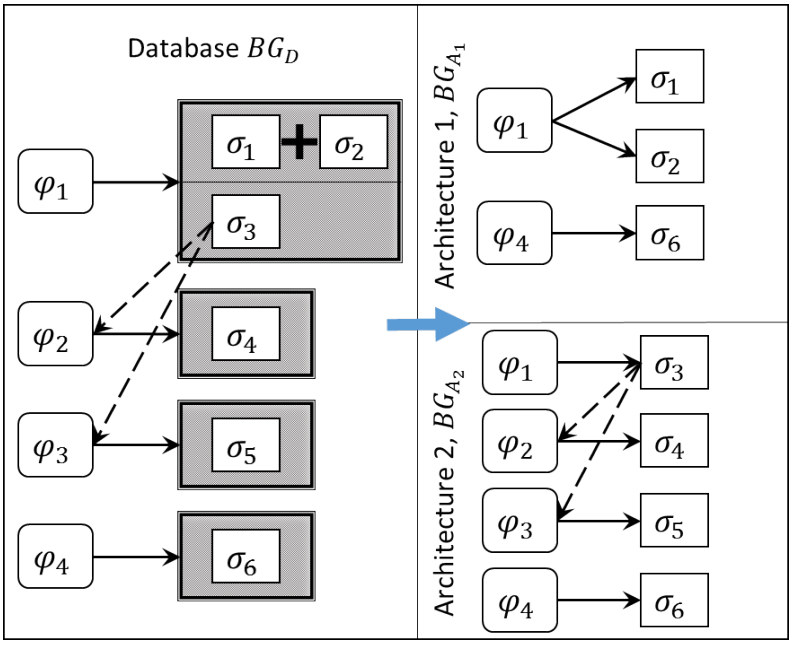


Figure 4.3: *Functional reasoning bipartite graphs.*

4.4.2.4 Inter-Domain Relations:

During the systems architecting process, the architects specify mapping relations (as discussed in Section 4.4.1) between elements of different domains. The relations between the elements from two different domains are referred to as inter-domain relations. For instance, the architect may first specify a requirement-to-function mapping relation between a requirement ($\in R$) and a function ($\in \Phi$), and then a function-to-solution mapping relation between the function ($\in \Phi$) and a solution ($\in \Sigma$). As these two relations are directly specified, these are referred to as direct relations. However, these two direct relations also imply an indirect mapping relation between the requirement ($\in R$)

and the solution ($\in \Sigma$). These indirect relations need to be identified and managed so that the architect is able to take an informed decision while modifying the architecture. For instance, if a performance requirement cannot be satisfied by a given architecture, then the architect would be able to determine which solutions (components) affect those performance requirements.

In the proposed framework, undirected graphs UG are employed to model the inter-domain relations. The UG is constructed by using the architect defined inter-domain relations (e.g. requirement-to-function mapping, $\rho \circ\!\!- \varphi$, function-to-solution mapping, $\varphi \circ\!\!- \sigma$, solution-to-function mapping (derived function) $\sigma \twoheadrightarrow \varphi$, requirement-to-solution mapping, $\rho \circ\!\!- \sigma$, etc.) The elements of the R, F, L and C (notional computational domain containing description of computational models and workflows) domains are represented by sets, R, Φ, Σ, and M , as shown by Equations 4.3, 4.4, 4.5 and 4.6, respectively.

$$R = \{\rho_1, \rho_2, ..., \rho_r\} \tag{4.3}$$

$$\Phi = \{\varphi_1, \varphi_2, ..., \varphi_f\} \tag{4.4}$$

$$\Sigma = \{\sigma_1, \sigma_2, ..., \sigma_c\} \tag{4.5}$$

$$M = \{\mu_1, \mu_2, ..., \mu_r\} \tag{4.6}$$

The inter-domain relations are expressed by $\mathscr{R}$. The relations between the elements of any two sets A and B can be represented by Equation 4.7, where, $A, B \in \{R, \Phi, \Sigma, M\}$.

$$A \mathscr{R} B = \{(a,b) | a \circ\!\!- b \wedge (a \in A \wedge b \in B) \wedge (A \neq B)\} \tag{4.7}$$

There are six types of inter-domain relations possible that exist between the elements of the sets, R, Φ, Σ , and M as follows: $M \mathscr{R} \Phi, \Phi \mathscr{R} \Sigma, \Sigma \mathscr{R} M, R \mathscr{R} \Sigma, \Phi \mathscr{R} M$, and $R \mathscr{R} M$. For instance, $R \mathscr{R} \Phi$ is a set of relations between requirements and functions, and a mapping relation between a requirement (ρ_i) and a function (φ_i) can be represented by $\rho_i \mathscr{R} \varphi_i \in R \mathscr{R} \Phi$ and,

$$R \mathscr{R} \Phi = \{(a,b) | a \circ\!\!- b \wedge (a \in R \wedge b \in \Phi)\} \tag{4.8}$$

All these relations ($\mathscr{R}$), obtained by combining all types of relations described above, are considered symmetric. For example, consider a relation $\rho_i \, \mathscr{R} \, \varphi_i$, where a requirement ρ_i is mapped/fulfilled by a function (φ_i); this also means the function is fulfilled by/mapped to the requirement. This symmetric relation is mathematically expressed in Equation 4.9. Similarly, other relations can also be defined this way.

$$\forall_{\rho_i \in R, \varphi_i \in \Phi} (\rho_i \, \mathscr{R} \, \varphi_i \in \mathscr{R}) \Rightarrow (\varphi_i \, \mathscr{R} \, \rho_i \in \mathscr{R}) \tag{4.9}$$

Furthermore, these relations are also transitive. For instance, when a requirement (ρ_i) is fulfilled by a function (φ_i), and the function (φ_i) is fulfilled by a solution (σ_i), it can be deduced from this that the solution (σ_i) fulfils the requirement (ρ_i), and the deduced relation is an element of the relations set,$\mathscr{R}$.

$$\exists_{\rho_i \in R, \varphi_i \in \Phi, \sigma_i \in \Sigma} (\rho_i \, \mathscr{R} \, \varphi_i \in \mathscr{R} \wedge \varphi_i \, \mathscr{R} \, \sigma_i \in \mathscr{R}) \Rightarrow (\rho_i \, \mathscr{R} \, \sigma_i \in \mathscr{R}) \tag{4.10}$$

Therefore, the inter-domain relations set, $\mathscr{R}$, defined over the architectural elements is symmetric and transitive in nature. Figure 4.4 illustrates inter-domain relations, where the architect specified relations are shown by solid lines, and the indirect relations are shown by dotted lines. During the architecting process, only some of the relations (of the above types) may be defined by the architect. However, there is a need to dynamically deduce the indirect ones because the knowledge of these is essential for the dependency analysis of the architectural elements. To obtain the indirect relations, a transitive closure graph (G^{TC}) of the dynamically generated initial graph (G^i) is obtained. G^i is extracted from the object model storing architectural information. Algorithms for tracing the direct and indirect relations within and between the RFLC domains are presented in the next section.

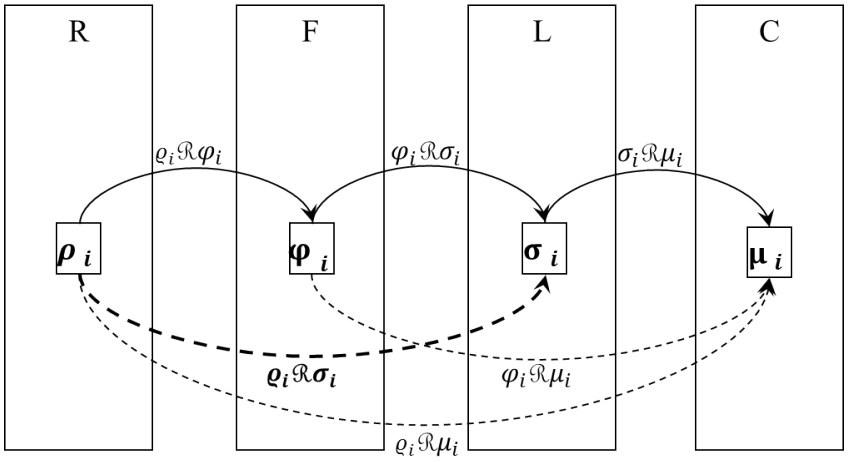


Figure 4.4: *Relations among elements of RFLC domains.*

4.5 Dependency Analysis

In this section, algorithms enabling traceability between domains are proposed. These algorithms are applied on the data structures described in Section 4.4.2. Three main algorithms are presented: the first one enables tracing functional-logical zigzagging, the second one allows inter-domain tracing, and the third supports tracing specifically within the computational domain. It should be noted that these algorithms are generic and not specific to any implementation framework. For testing purposes, the algorithms are implemented using an object oriented programming language, C#, in in-house architecting tool, AirCADia Architect as described in Appendix C.

4.5.1 Tracing Between Functional and Logical Domains

In order to trace the dependency between functions and solutions of a given function reasoning model, a traversal algorithm based on a Depth First Search (DFS)[88] procedure has been employed. If the architect wishes to modify the architecture by adding, removing or substituting a function or solution, the framework would assist by highlighting the impact of the desired change. This is illustrated in Figure 4.5, where the bipartite graph for the complete database is shown in the middle and the logical and functional flow views on the left and right, respectively. If the architect wishes to see the implication of removing solution σ_4 from the logical flow view, the corresponding

dependency (traversal) can be obtained by employing Algorithm 4.1 and Algorithm 4.2, as highlighted by the red arrows. This allows to identify the affected functions which are highlighted in grey.

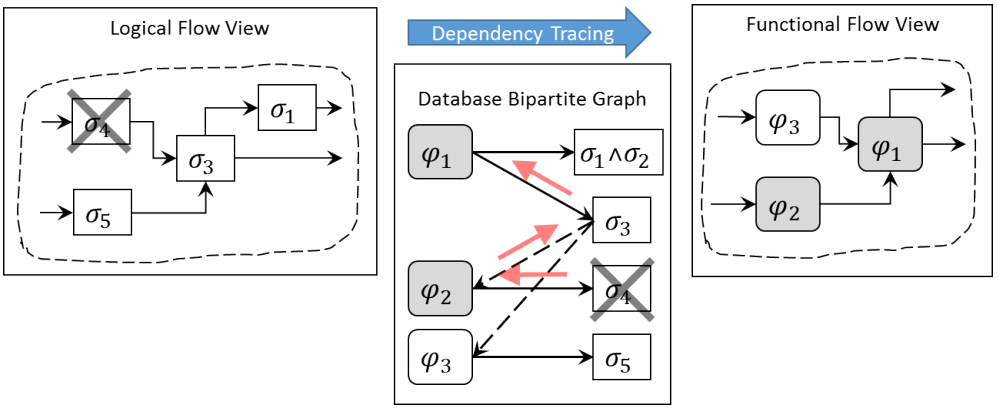


Figure 4.5: *Dependency tracicng between functional and logical domains using bipartite graph.*

The first step in Algorithm 4.1 is to populate and update the bipartite graph, *A_BG* (see implementation in Figure C.13). Here, *A_BG* is updated for every user action involving addition/deletion of solution/function, establishing/deleting mapping or derived relations. Then, the second step is to determine the dependency of the requested element 'e' as described in Algorithm 4.2.

Algorithm 4.1: Part 1: Bipartite graph generation.

Input : System architecture (A), User activity (UA), and A directed bipartite graph, A_BG^i (i.e. $(G(\Phi,\Sigma,E))$) storing the *functional reasoning* knowledge in F and L views of the architecture

Output: An updated directed bipartite graph, A_BG

1 $A\_BG = A\_BG^i = G(\Phi,\Sigma,E)$; `// Φ - a set of functions, Σ - a set solutions, and E - a set of edges between functions and solutions`
2 **if** *UA is 'Addition of a solution in the logical view'* **then**
3 | Add corresponding solution node (S) in A_BG;
4 **else if** *UA is 'Addition of a function in the functional view'* **then**
5 | Add corresponding function node (F) in A_BG;
6 **else if** *UA is 'Deletion of a solution from the logical view'* **then**
7 | Delete corresponding solution node (S) from A_BG;
8 **else if** *UA is 'Deletion of a function from the functional view'* **then**
9 | Delete corresponding function node (F) from A_BG;
10 **else if** *UA is 'Addition of a mapping relations between F and S'* **then**
11 | Add directed link from corresponding node F to S in A_BG;
12 **else if** *UA is 'Addition of a derived relations between F and S'* **then**
13 | Add directed link from corresponding node S to F in A_BG;
14 **else if** *UA is 'Removal of a mapping relations between F and S'* **then**
15 | Delete directed link from corresponding node F to S of A_BG;
16 **else if** *UA is 'Removal of a derived relations between F and S'* **then**
17 | Delete directed link from corresponding node S to F of A_BG;
18 Return A_BG ;

Algorithm 4.2: Part 2: Dependency tracing between functional and logical domains.

Input : A directed bipartite graph, A_BG (i.e. $(G(\Phi,\Sigma,E))$) storing the *functional reasoning* knowledge in F and L views of the architecture (Here, Φ is a set of vertices of type function, Σ is a set of vertices of type solution, and E is a set of directed edges between the vertices from the above sets.), Architectural element (function or solution) - e.

Output: A list, L, of affected functions/solutions.

```
1  L := {} and S := {};
2  visited = {}; // empty dictionary 'visited' to store the status of elements (visited
   or not).
3  for vertex u in A_BG do
4      set visited[u] := false;
5  Add 'e' in S;
6  while S is not empty do
7      Remove last element in 'S' and assign it to u;
8      if not visited[u] then
9          visited[u] := true;
10     Add u in L;
11     for unvisited neighbours w of u do
12         Add w in S; // here, the graph is traversed in the opposite direction of
           the edges of A_BG.
13 Return L;
```

4.5.2 Tracing Across the Domains

A transitive closure algorithm is employed to identify all indirect relations between elements of different domains of the systems architecture. For an initial graph, $G^i = (V, E)$, V is the vertex set and E is the edge set. The transitive closure graph of G^i is a graph, $G^{TC} = (V, E^+)$, such that for all vertices pair, (v_i, v_j) in V, there is an edge (v_i, v_j) in E^+, if and only if there is a non-null path from v_i to v_j in G^i [88]. In the proposed framework, the initial graph G^i is constructed with direct relations only. It is important to note that in the framework implementation, G^i is automatically generated 'behind the scenes' from the specified elements and relations. After applying the transitive closure on G_i, the resulting graph, G^{TC}, contains both direct and indirect relations. For illustration, an initial graph (G^i) containing four vertices ($x1, x2, x3$, and $x4$) and three edges (($x1, x2$), ($x2, x3$)), ($x3, x4$)) is shown in Figure 4.6 (top). The graphs are also shown in the form of the adjacency matrices as shown on the right of the figure. The obtained transitive closure graph (G^{TC}) containing extra three edges (indirect) (($x1, x3$), ($x2, x4$) and ($x1, x4$)) as paths exist between $x1$ to $x3$, $x2$ to $x4$, and $x1$ to $x4$ respectively (as shown at the bottom of the figure).

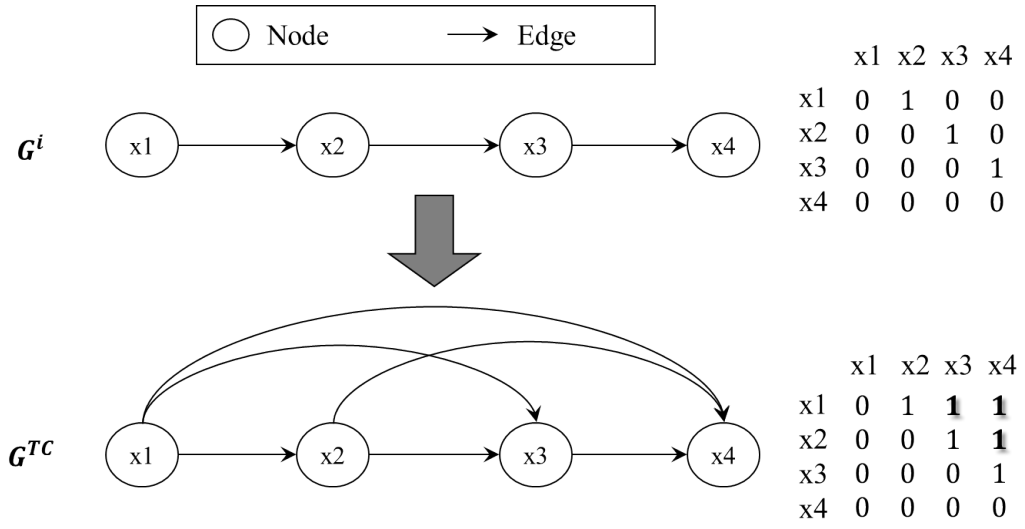


Figure 4.6: *Transitive closure of the graph.*

Algorithm 4.3: Transitive closure graph generation.

Input : System architecture (A), i.e., element sets R, Φ, Σ and M of R, F, L and C domains

Output: Transitive closure graph, G^{TC} which stores all (direct and indirect) relations

1 $E = R \cup \Phi \cup \Sigma \cup M$;
2 $m = |E|$;
3 Create initial graph storing direct relations, adjacency matrix, $G^{i}_{m \times m}$;
4 $G^{i}_{m \times m} = I_{m \times m}$;
5 **for** $i = 1$ *to* $i \leq m$ **do**
6 **for** $j = 1$ *to* $j \leq m$ **do**
7 **if** $E[i]$ *is associated to* $E[j]$ **then**
8 $G^{i}_{m \times m}[i, j] = 1$;
9 **end**
10 **end**
11 **end**
12 $G^{TC} = G^{i}_{m \times m}$;
13 **for** $i = 1$ *to* $i \leq m$ **do**
14 **for** $j = 1$ *to* $j \leq m$ **do**
15 **if** $G^{TC}[i, j] = 1$ **then**
16 Perform element-wise 'OR' operation on i^{th} row of G^{TC} and j^{th} row of G^{TC} and assign it to $V_{1 \times m}$;
17 Assign $V_{1 \times m}$ to i^{th} row of G^{TC};
18 **end**
19 **end**
20 **end**
21 Return G^{TC};

Algorithm 4.4: Inter-domain dependency tracing.

Input : System architecture (A), i.e., element sets (R, Φ, Σ and M) of R, F, L and C domains respectively; an entity, 'e' whose relation with other entities (of other domain) is to be traced; and transitive closure graph, G^{TC}.

Output: Related entities list, L.

```
1  E = R ∪ Φ ∪ Σ ∪ M ;
2  m = |E|;
3  for i = 1 to i ≤ m do
4      if E[i] = e then
5          for j = 1 to j ≤ m do
6              if G^TC[i, j] = 1 and i ≠ j then
7                  Add E[j] in L;
8              end
9          end
10         Break;
11     end
12 end
13 Return L;
```

4.5.3 Tracing Within the Computational Domain

This algorithm supports the tracing of entities within the computational domain It obtains the dependency of a given parameter (whose dependency is to be traced) on other parameters. For this, a relevant computational workflow (a directed bipartite graph (see Chapter 5)) is employed.

Algorithm 4.5: Tracing dependency within computational domain.

Input : Workflow, W, a directed bipartite graph with two types of nodes (model (M) and parameter (V)) and a parameter, 'p' whose relation with other parameters (within the domain) is to be traced.

Output: Related parameters list, L.

```
1  W = G(M, V);
2  L := {};
3  S := {};
4  for vertex u in W do
5      set visited[u] := false;
6  Add 'e' in S;
7  while S is not empty do
8      Remove last element in 'S' and assign it to u ;
9      if not visited[u] then
10         visited[u] := true;
11     Add u in L;
12     for unvisited neighbours w of u do
13         Add w in S;
14 K = {};
15 for i = 1 to i ≤ |L| do
16     if L[i] is not a solution parameter then
17         Add L[i] to K;
18 for i = 1 to i ≤ |K| do
19     Remove K[i] from L;
20 Order the parameters of the L as per their sensitivity to parameter p;
21 Return ordered L;
```

4.6 Summary

Presented in this chapter is a novel method for dependency analysis of architectural elements. The method is supported by different data structures that describe the architectural information and a set of algorithms which are applied on the former to trace the dependency.

An object model has been extended from the work by Guenov et al.[44], to include requirements and the notion of a computational domain. More details on the software implementation of the object model are given in Appendix C. This software is employed for further testing of the proposed approach.

Tree and directed graphs are employed to store hierarchical and flow information of the architectural views. An undirected graph is used to store inter-domain relations. Two separate bipartite graphs are proposed for storing *functional reasoning* knowledge. One is used to capture reasoning knowledge in the particular architecture, i.e. functional and logical views, and another is employed to store similar knowledge of different architectures. These data structures are updated behind the scene on each of the user actions on the architecture. In case of inter-domain relations, an initial graph is generated to store the user-specified (or direct) relations, and using these relations, a graph storing both direct and indirect relations is produced.

Three main algorithms are proposed for tracing dependency: 1) between functional and logical domains, 2) across the domains, and 3) within the computational domain. These algorithms are employed in conjunction to find the dependency information of an architectural element.

CHAPTER 5

Architecture Assessment

In the previous chapter, the enablers for the architecture definition process were described. In this chapter, the architecture assessment enablers are described. Usually, after creating the architecture, the architect wishes to assess the effect of the architectural changes on the system level performance. The architecture assessment consists of sizing the sub-systems and then finding system-level performance.

5.1 Overview of Architecture Assessment Process

During Architecture Design Space Exploration (ADSE), the architecture definition and assessment processes are sequentially performed for several architectures. Once the system architecture is defined, the next step is to assess performance at the system level. The assessment process can be seen as a sequence of three steps as shown in Figure 5.1.

The process takes the system architecture's requirement and logical views as input. The process starts with the orchestration of the sub-systems for their sizing, followed by the

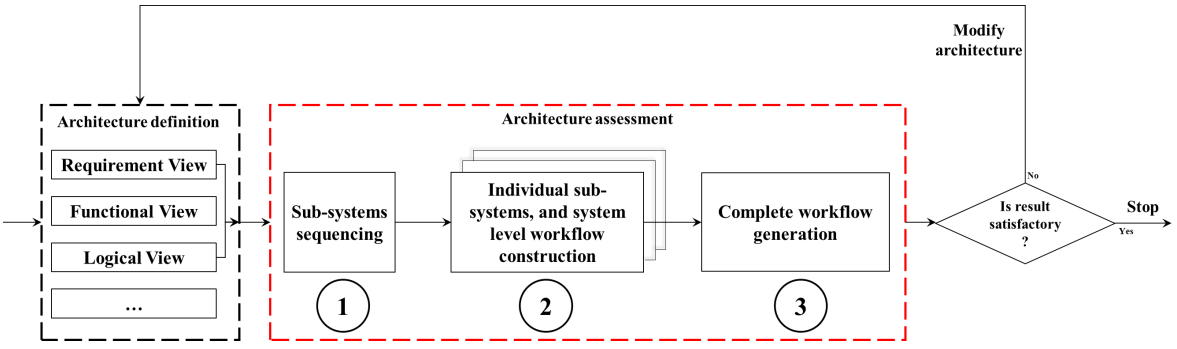


Figure 5.1: *Steps involved in the architecture assessment process.*

construction of the sub-systems and system workflows. Finally, a complete computational workflow, combining the sub-systems sequence and corresponding workflows, is generated and executed. The results are given back to the architect, and the process is repeated, if needed.

The rest of the chapter describes the support developed for architecture assessment in line with the process in Figure 5.1. Firstly, the theoretical foundation behind the assessment process, given the requirement and logical views of the architecture as input, is outlined. Then, Section 5.3 elucidates the method offered for automating the first step, i.e., sequencing the sub-systems. An enabler proposed for the construction of the workflow, supporting step-2 of the assessment process, is described in Section 5.4, followed by details on the generation of the complete workflow in Section 5.5. Finally, a summary of the developed enablers supporting the architecture assessment process is presented in Section 5.6.

5.2 Theoretical Foundations

This section outlines the theoretical foundation behind the architecture assessment process. As mentioned above, the proposed architecture assessment methodology takes a system architecture's logical and requirements views, as input. The system architecture can be represented in a hierarchical structure as shown in Figure 5.2. Here, a system is an assembly of sub-systems, and these sub-systems are connected to each other to describe a flow of entities such as, material, energy and signal, between them. In turn, a sub-system is composed of components, and connections exist among them which

describe the flow of the above entities. Each of the entities is described by a set of parameters, and for each of the sub-systems/components, some entities flow in and out of them. The rest of this section describes the notations employed to describe the logical view of the architecture.

A system is composed of its sub-systems (SSs). Mathematically, this can be represented by a set, S, of sub-systems (SS) as shown in Equation 5.1.

$$S = \{SS^1, SS^2, SS^3, ...\} \tag{5.1}$$

In Figure 5.2, three representative sub-systems, SS^k, SS^i, and SS^l are showed. Here, a superscript represents a sub-system index.

Similarly, a sub-system is composed of its components. The i^{th} sub-system,SS^i, can be represented by Equation 5.2. For example, the first component of the i^{th} sub-system is denoted by C_1^i. Here, a subscript represents a component index.

$$SS^i = \{C_1^i, C_2^i, C_3^i, ...\} \tag{5.2}$$

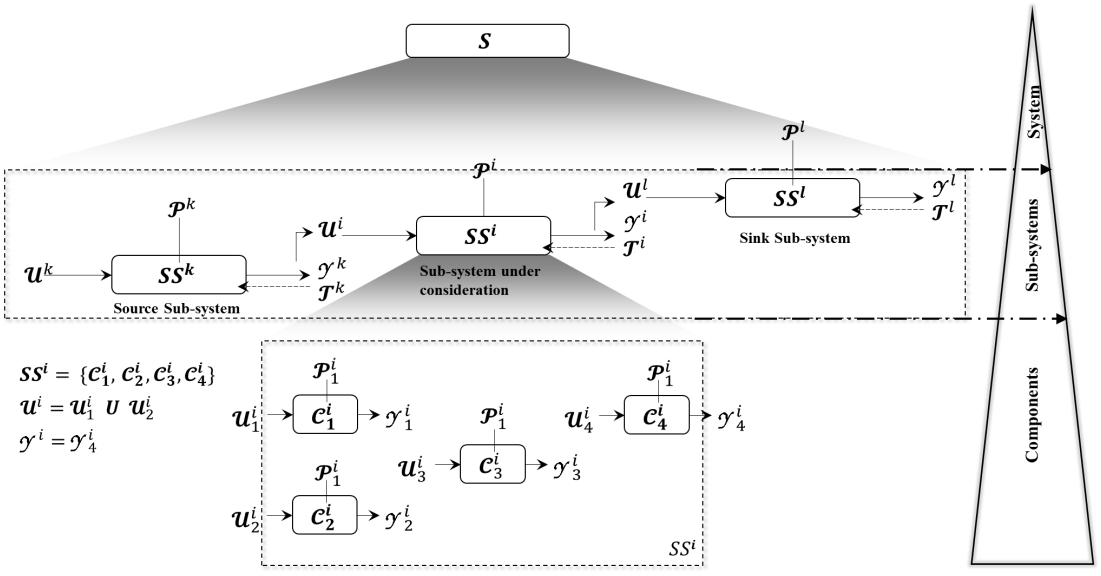


Figure 5.2: *Formulation of a sub-system sizing.*

A component (or solution)* in the logical view is assigned ports through which it is connected to other components. The port types are classified according to the physical entities "flowing" through the connections, for example, material, energy and signal [24] [23]. The connections also define the direction of the physical flow among the components. It is assumed that connections can only exist between the same types of ports.

Each port, depending on its type, has some parameters associated with it. For instance, the electrical (energy) port is associated with voltage (effort) and current (flow) parameters, and a power conjugate, which is the product of the above two. For a component as shown in Figure 5.3, the sets of parameters of its input and output ports are represented by $\mathcal{U}_j^i$ and $\mathcal{Y}_j^i$, as shown in Equation 5.3 and 5.4, respectively. A component has its own parameters, referred to as component-parameters, represented by the set, $\mathcal{P}_j^i$ as shown in Equation 5.5. Thus, in total, a component possess three sets of parameters: 1) a set of parameters associated to its input ports, $\mathcal{U}_j^i$, 2) a set of parameters associated to its output ports, $\mathcal{Y}_j^i$, and 3) a set of parameters associated to the component itself, $\mathcal{P}_j^i$. For example, a compressor component may have three sets of parameters, i.e. parameters associated with: two input ports (one for air inlet and the other for required energy to compress air), an output port (for air outlet), and the compressor own parameters. The set $\mathcal{U}$ is comprised of five parameters, associated to the two input ports, i.e. pressure (P), temperature (T) and mass flow rate ($\dot{m}$) of the air at the inlet of the compressor, and torque (τ) and rotational velocity (ω) at the input energy port. The set $\mathcal{Y}$ contains three parameters, associated with the output air-port, i.e. pressure (P), temperature (T) and mass flow rate ($\dot{m}$) of the air at the outlet. Finally, the set $\mathcal{P}$ may contain parameters associated with the compressor components, i.e. compression ratio (γ) and weight (W).

$$\mathcal{U}_j^i = \{u_{j,1}^i, u_{j,2}^i, u_{j,3}^i, ...\} \tag{5.3}$$

$$\mathcal{Y}_j^i = \{y_{j,1}^i, y_{j,2}^i, y_{j,3}^i, ...\} \tag{5.4}$$

$$\mathcal{P}_j^i = \{p_{j,1}^i, p_{j,2}^i, p_{j,3}^i, ...\} \tag{5.5}$$

*It should be noted that terms, 'solution' and 'component', are used interchangeably unless 'component' refers to a level in the system decomposition (hierarchy).

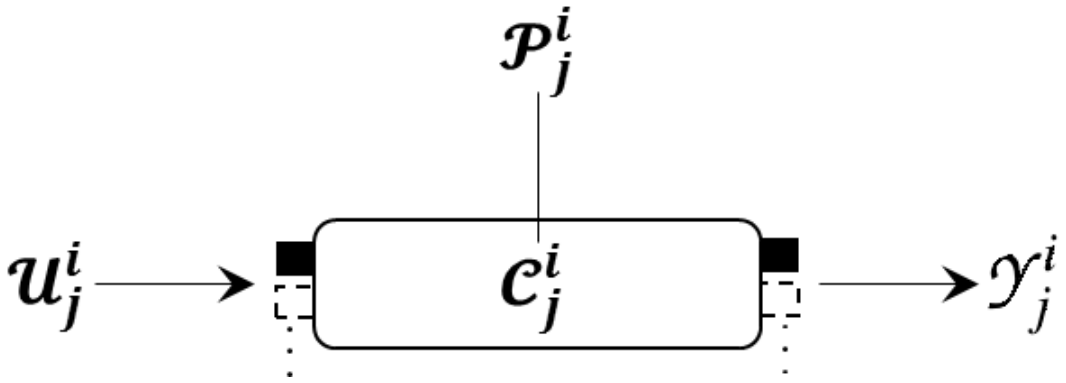


Figure 5.3: *Component and its associated sets.*

For each component of a sub-system, there exist one or more computational models to simulate its intended behaviour (i.e. the function of the component). A component can have multiple behaviours. However, the intended behaviour is the component's intended action which it performs within the system. For instance, an electric motor has several behaviours such as producing vibration or noise, converting electrical energy into heat, and converting electrical energy into rotational energy. However, the only intended behaviour of the motor is converting electrical energy into rotational energy. In this research, the considered computational models associated with a component are those that simulate the intended behaviours (or functions) of the component. However, the proposed approach is generic, and therefore, given the associated (unintended) behaviours, their corresponding computational models and associated requirements, the models can be incorporated in the sizing workflows.

Models can have multiple inputs and outputs. The component $\mathcal{C}_j^i$ of the logical view may have more than one computational model associated with it, represented by a set, $\mathcal{M}_j^i$ (read as a set of models associated with j^{th} component of sub-system i), as shown in Equation 5.6. These computational models describe the function of the associated component by mathematical relationships among the variables (parameters of the component). In addition to these necessary behavioural models, those calculating other aspects of the component such as cost, weight etc. can also be included in this set.

$$\mathcal{M}_j^i = \{m_{j,1}^i, m_{j,2}^i, m_{j,3}^i, ...\} \tag{5.6}$$

A sub-system also holds similar sets to those of a component, as described in Equation

5.7, 5.8, 5.9 and 5.10. In addition, it possesses a set of target values. This set stores the required values of variables in the set, $\mathcal{Y}^i$. The model set associated with the sub-system also contains dynamically generated sub-system variable aggregation models (DGM^i) (for more details see Section 5.4.1).

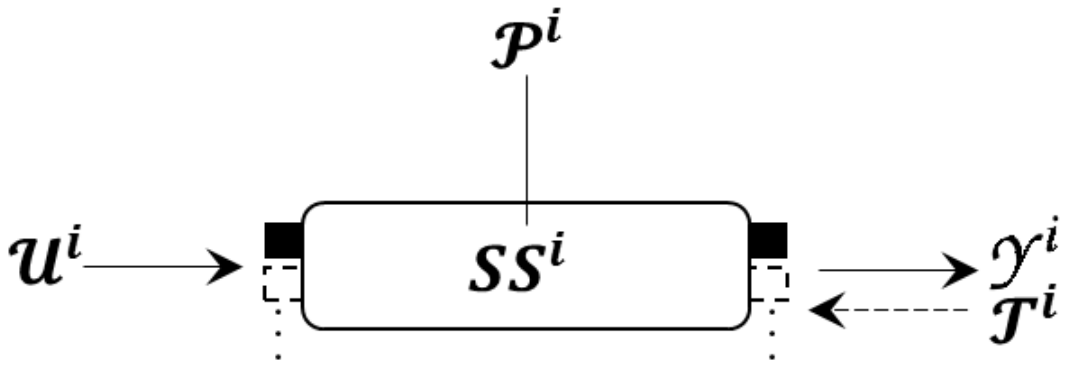


Figure 5.4: *Sub-system and its associated sets.*

$$\mathcal{U}^i = \{u_1^i, u_2^i, u_3^i, ...\} \tag{5.7}$$

$$\mathcal{Y}^i = \{y_1^i, y_2^i, y_3^i, ...\} \tag{5.8}$$

$$\mathcal{P}^i = \{p_1^i, p_2^i, p_3^i, ...\} \tag{5.9}$$

$$\mathcal{M}^i = \{m_1^i, m_2^i, m_3^i, ...\} \tag{5.10}$$

A sub-system level logical view of the system can be obtained by grouping corresponding sub-system components. The logical connections among these groups (or sub-systems) can be deduced from the logical connections between the respective components of these sub-systems. For a given logical connection between two sub-systems, one of these is a source and other is a sink, depending on the direction of the physical entity flowing through the connection. For instance, for the sub-system, SS^i, under consideration in Figure 5.2 SS^k, and SS^l are source and sink sub-systems, respectively. SS^i is both a source and a sink for SS^l and SS^k, respectively. More details on how source-sink relations among the sub-systems are formed are given in Section 5.3. It should be mentioned here that the sink is sized before the source sub-system, so that the inputs required by the sink system become targets on the source sub-system. Thus, in addition to the sets expressed in Equations 5.7, 5.8, 5.9 and 5.10, the i^{th} sub-system

possesses a set of target values which are required (determined by and calculated at the sink sub-system) on the output variable set, $\mathcal{Y}^i$ (in case of i^{th} sub-system). These values are calculated at the sink sub-system. For the sub-system, SS^i, depicted in Figure 5.2, these are calculated at the sub-system SS^l as expressed in Equation 5.11. That is, as the sink sub-system SS^l is sized first, the values of the variables of the set, $\mathcal{U}^l$, are known while sizing sub-system, SS^i. Therefore, these values, i.e. $\mathcal{T}^i$, become target on the variables of the set, $\mathcal{Y}^i$.

$$\mathcal{T}^i = \{t_1^i, t_2^i, t_3^i, ...t_{nt}^i\} \tag{5.11}$$

Thus, all the variables that are associated with a sub-system are stated by a set as shown in Equation (12).

$$\mathcal{V}^i = \{v_1^i, v_2^i, v_3^i, ...\} = \mathcal{U}^i \cup \mathcal{Y}^i \cup \mathcal{P}^i \cup \bigcup_{j=1}^{n} \mathcal{U}_j^i \cup \bigcup_{j=1}^{n} \mathcal{Y}_j^i \cup \bigcup_{j=1}^{n} \mathcal{P}_j^i$$

As the design progresses, the number of parameters employed to describe the system (and its components) also increases. Although the methodology proposed here does not restrict the use of particular models, the research focus is on steady state, low fidelity models as during early stage system design, the design information available about the system to be designed is limited. In this work, the low fidelity models considered describe the system (its components) with limited design information. Additionally, their execution time is acceptable to be used in early stage system design.

Similarly, the system level (of the system) would have separate computational models that find the system level performance. These models may need input from the (lower level) sub-systems i.e. parameters from the sub-systems may be input to the model(s) at the system level. For example, the 'propulsion system' is one of the aircraft sub-systems, and the specific fuel consumption (sfc) parameter is associated with the engine component of this sub-system. However, the sfc may be employed at the system level as input to the range calculation. Furthermore, some of the system level parameters could be an aggregation of parameters from the lower levels i.e. parameters of the sub-systems or components. For instance, the system weight parameter (at the system level) is the sum of the weights of all sub-systems.

Depending on the available design information, the design scenarios/problems can be broadly classified into 'synthesis' and 'analysis' as shown in Table 5.1. Here, $\mathcal{U}$, $\mathcal{Y}$, $\mathcal{P}$ are the sets associated with a sub-system or a component, and $\mathcal{UY}_c$ and $\mathcal{P}_c$, are constraints on variables in the set $\mathcal{U}$ and $\mathcal{Y}$, and $\mathcal{P}$, respectively. The tick symbol in the table denotes that design information associated with the column (parameter set) is available; whereas, the question mark represents the unknown design information (set corresponding to the column) in the respective scenario.

TABLE 5.1: Design scenarios or problems.

Scenarios	$\mathcal{U}$	$\mathcal{Y}$	$\mathcal{P}$	$\mathcal{UY}_c$	$\mathcal{P}_c$	Given	Find	Purpose
Synthesis	~	✓	?	✓	✓	$\mathcal{U}, \mathcal{Y}, \mathcal{UY}_c, \mathcal{P}_c$	$\mathcal{P}$	Sizing
Analysis	~	?	✓	✓	✓	$\mathcal{U}, \mathcal{P}, \mathcal{UY}_c, \mathcal{P}_c$	$\mathcal{Y}$	Performance Evaluation

A system sizing task encompasses sizing of its parts. Here, this task is sub-divided into the sub-systems' sizing tasks because of the following reasons.

- To deal with the complexity (due to a large number of components involved) of a system and computations involved in sizing.
- To mimic an industrial set-up where computational models are developed by different teams for their respective sub-systems.
- To try different fidelity models for a sub-system during sizing of the system without requiring modification in the computational set-up of the other sub-system sizing tasks.
- To execute independent sizing tasks in parallel.

The next section describes the enabler, which supports achieving the first step of the assessment process, i.e. sequencing of the sub-systems for their sizing.

5.3 Step-1: Sequencing of the Sub-systems

This step involves orchestration of the sub-systems for sizing, depending on their interdependency. The method uses the source-sink approach as a basis for devising the sizing

sequence. A source-sink relation between any two sub-systems is created by connecting relevant ports (Figure 5.5), for example, connecting the mass flow ports of the engine (bleed air) (sub)system and the de-icing (sub)system. Depending on the flow direction of the connection, one of the associated sub-systems is the source and other is the sink, as shown in Figure 5.5. The rationale behind the sequencing of the sub-systems is that the sink sub-system should be sized before the source sub-system. That is, the sink sub-system sets targets for the source sub-system during the sizing process.

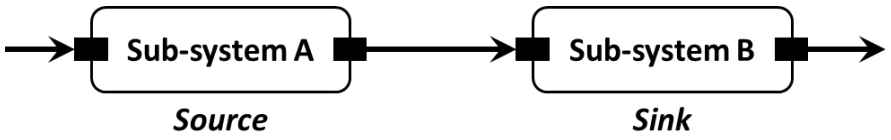


Figure 5.5: *The source-sink relation between the sub-systems.*

5.3.1 Source-sink DSM

Following this approach, a source-sink Design Structure Matrix (DSM) of the logical view at the sub-system level is created. The DSM is a square matrix where each row and column represents a sub-system. The matrix elements store the source-sink relations between the sub-systems. If a non-diagonal element is marked with '1', then the sub-system associated with the corresponding row is a source, whereas the sub-system associated with the column is a sink. Figure 5.6 shows a generic example of a sub-system (SS) level representation of the system logical view. The source-sink DSM of the logical architecture of Figure 5.6 is shown in Figure 5.7.

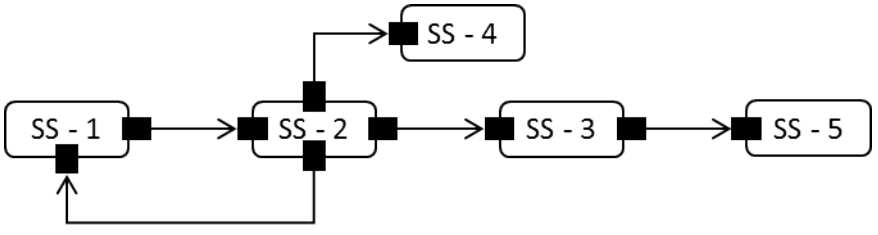


Figure 5.6: *Sub-system (SS) level logical view of a system.*

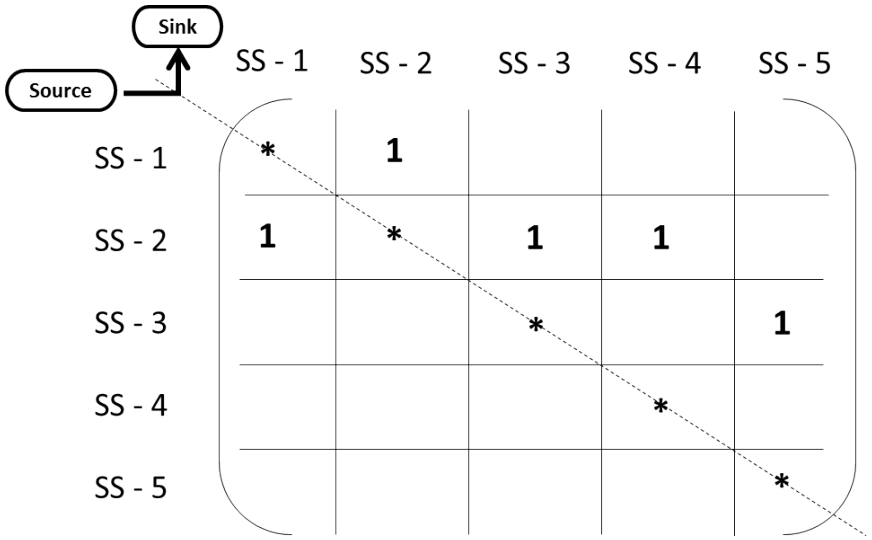


Figure 5.7: *Source-sink DSM.*

5.3.2 DSM Sequencing

Given the source-sink DSM, the sizing sequence of the sub-systems is obtained by first finding the coupled sub-systems, also called Strongly Connected Components (SCCs) and creating a new matrix in which these SCCs are represented as single entries. This new matrix is, then, transformed into a Lower Triangular Matrix (LTM) form. Following that, the single SCC entries in the LTM are expanded back to their original state. The resulting matrix is referred to as the Sequenced DSM. Then, the square sub-matrices formed at the diagonal of this sequenced matrix are utilized to extract the sequence. There are three types of the sub-matrices that can be formed at the diagonal of the matrix. These are shown in Table 5.3, where it is assumed that the sub-matrix is of dimension $p \times p$, where $p \leq n$ ($DSMsize$). The first type is a diagonal matrix. The second type is a lower-triangular matrix, whereas the third type is a matrix with at least one non-zero element above the diagonal.

TABLE 5.2: Types of sub-matrices formed along the sequenced DSM diagonal.

Sub-matrix Type	Mathematical Description	
Type - 1	$\forall_{i \leq p} \forall_{j \leq p} x_{ij} = 0$	$i \neq j$
Type - 2	$[\forall_{i \leq p} \forall_{j \leq i} x_{ji} = 0] \wedge [\exists_{i \leq p} \exists_{j \leq i} x_{ij} \neq 0]$	$i \neq j$
Type - 3	$\exists_{i \leq p} \exists_{j \leq p} x_{ij} \neq 0$	$i \neq j$

The sequenced DSM obtained for the source-sink DSM of Figure 5.7 is shown in Figure 5.8. The figure also shows the three types of the sub-matrices that are formed at the diagonal. These sub-matrices are shown by solid, dashed and chain-lined rectangles, respectively.

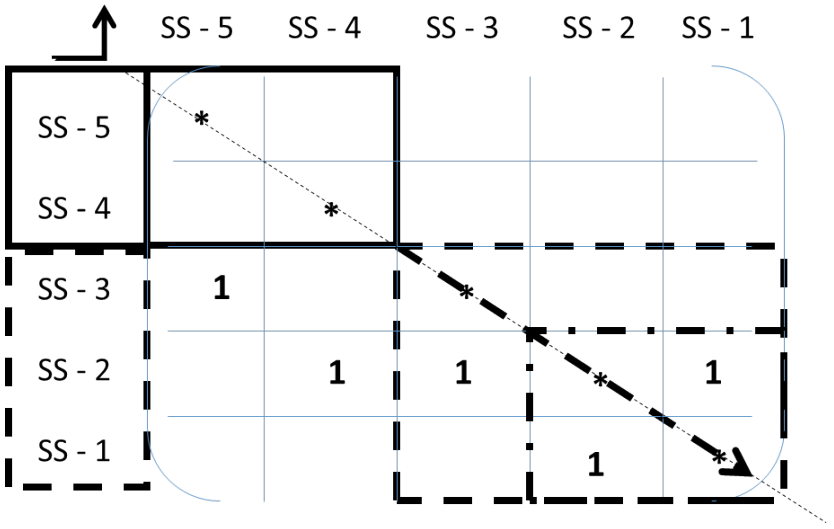


Figure 5.8: *A sequenced DSM and associated sub-matrices at the diagonal.*

The sizing sequence of the sub-systems is determined by the type of the sub-matrices along the sequenced DSM main diagonal. Table 5.3 gives the sub-system sizing sequence as per the sub-matrices of the sequenced matrix in Figure 5.8.

TABLE 5.3: Sub-systems sequence as per the sub-matrix types.

Sub-matrix Type	Designation in Figure 5.8	Associated Sub-systems (SS)	Sizing Sequence
Type - 1	Solid-lined rectangle	SS-5 and SS-4	Parallel
Type - 2	Dash-lined rectangle	SS-3, and (SS-2 and SS-1)	Sequential
Type - 3	Chain-lined rectangle	SS-2 and SS-1	Coupled (Iterative)

The execution sequence of the sub-system sizing tasks can be divided into stages for convenience. In particular, this staging can facilitate project and process management practices and enable appropriate allocation of computational resources, including parallel computation where possible.

The execution stages of the sizing tasks for the sub-systems in Figure 5.6, considering the associated sequenced DSM in Figure 5.8, are shown in Figure 5.9. The stages are established by tracing the sub-matrices along the sequenced DSM-diagonal from the top row to the bottom row. Here, SS-4 and SS-5 are associated with Type-1 sub-matrix; therefore, the sizing tasks associated with these sub-systems can be executed in parallel. Sub-systems SS-1 and SS-2 are contained in a SCC and need to be executed iteratively. Furthermore, the tasks (SS-4 and SS-5), SS-3, and the SCC (SS-1 and SS-2) are required to be executed sequentially because together they are associated with Type-2.

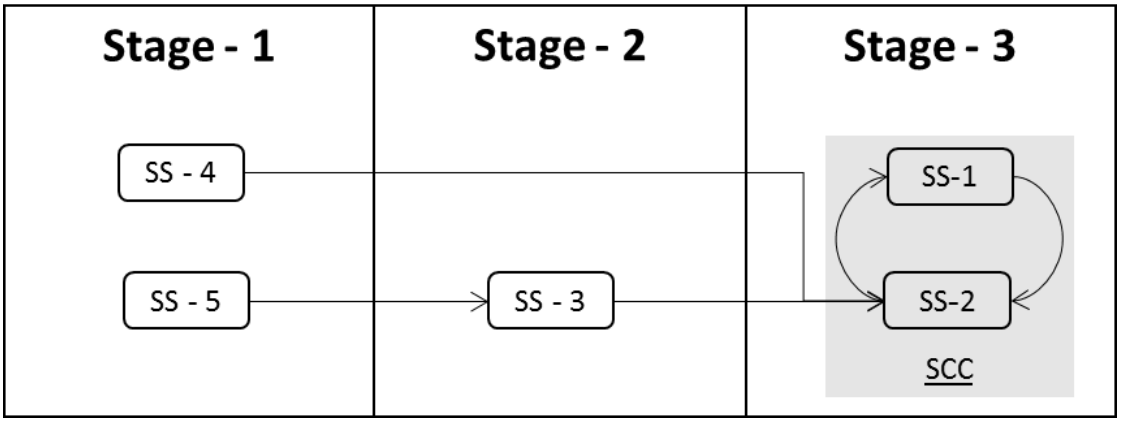


Figure 5.9: *Sub-systems sizing sequence.*

The sequencing method outlined above utilizes the following algorithms. The algorithm proposed by Xiao and Fei[89] is utilized to obtain a reduced DSM (with "lumped" Strongly Connected Components), to convert it into a Lower Triangular Matrix (LTM) and to obtain the sub-system sizing sequence. The algorithm identifies any Strongly Connected Components (SCCs) in the DSM and then reduces the DSM by aggregating the SCCs into single elements. The algorithm proposed by Tang et al.[90] is employed to obtain the SCCs of the original DSM.

Let's consider a system which contains m number of sub-systems (SS). Then the source-sink DSM is of dimension $m \times m$. Let 't' be the number of SCCs in the DSM. The SCC set is described as below.

$$S_{scc} = \{SCC_1, SCC_2, SCC_3, ..., SCC_t\} \tag{5.12}$$

Each of the SCCs, ($scc \in S_{scc}$) is associated with two or more elements of the DSM - $DSM_{m\times m}$, and an element cannot be part of more than one SCC of the set S_{scc}. If the number of elements in the i^{th} SCC (SCC_i) is given by $|SCC_i|$, then the total number of elements of $DSM_{m\times m}$ which are associated with SCCs, represented by z, is obtained.

$$z = \sum_{i=1}^{t} |SCC_i| \tag{5.13}$$

Considering each SCC as a single element, the original DSM is reduced to a matrix D^c of dimension $k \times k$, where $k = m - z + t$. To obtain the sizing sequence from the reduced DSM, an algorithm based on a theorem proposed by Xiao and Fei[89] is utilized. An illustrative example demonstrating application of the theorem on the source-sink DSM of Figure 5.7 is given in Appendix D.

In summary, the algorithm obtains the execution stages, by identifying empty rows in the reduced DSM. To achieve this, an initial k-dimensional column vector containing all elements equal to 1 is created, and multiplied by the reduced DSM. The resulting product is a column vector. If the vector contains '1's, the (sub-systems at) rows corresponding to those indices can be executed in parallel, i.e. in a single stage. The sub-systems (associated with the empty row) identified in each of the iteration-steps correspond to one of the stages in the sequence. Once the elements associated with the rows are placed in a certain stage, then, these rows and their corresponding entries in the DSM are ignored in the next iteration. This is achieved by setting the next step-column vector the following way: zeros corresponding to the covered rows and '1's corresponding to uncovered rows (or non-zero elements). The process is repeated until there are no more non-zero elements. It should be noted that the number of stages is not fixed and is not dependent on the number of sub-systems. As explained above, the number of stages is determined by the connectivity of the sub-systems, which is obtained from the logical view of the systems architecture.

5.4 Step-2: Construction of Workflows

In this step, computational workflows for the sub-systems and system level are obtained. First, individual sub-system workflows are produced, followed by construction of a system level workflow. During the process, aggregation and/or logical connections' (of a logical view) models are generated. In addition to these models, individual sub-system components and system level models are imported. A procedure (described later in Section 5.4.2) for constructing a workflow is repeatedly called to construct an individual workflows and should not be confused with the 'Step-2: Construction of Workflows' which is second step of the assessment methodology (see Figure 5.1). The tasks of 'Step-2' are depicted in the flowchart shown in Figure 5.10. A detailed explanation of the significant steps in the flowchart is given in the following sub-sections, while each step is briefly described below.

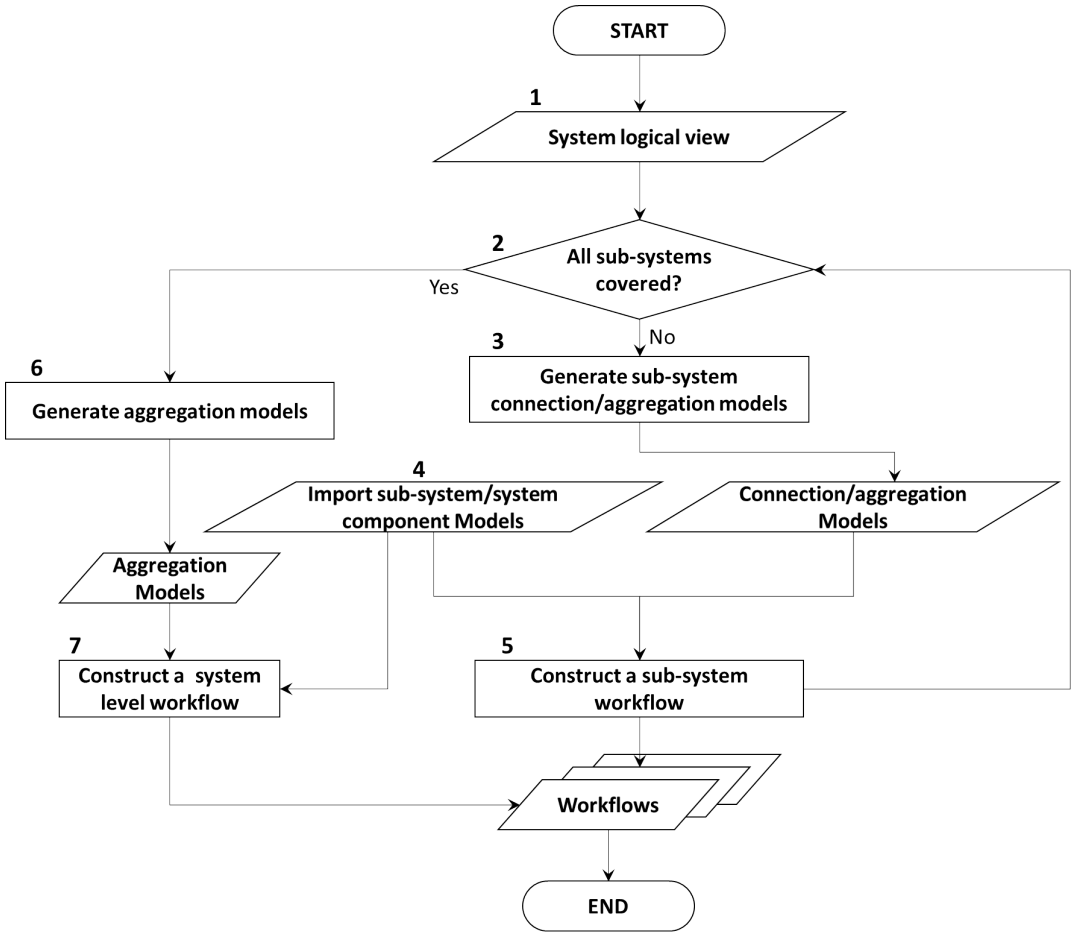


Figure 5.10: *Step-2: Construction of workflows.*

START

1. **System logical view:** The system logical view is taken as input. This is employed to access the sub-systems and, their components and logical connections among them.

2. **Condition - all sub-systems covered?:** This step identifies if all the sub-systems' workflows are created. If the algorithm identifies that there are still sub-systems for which workflows are not yet constructed then proceed to next step; otherwise, proceed to step-6 of the flow chart.

3. **Generate sub-system connection/aggregation models:** This step generates connection models among the components of the sub-system and aggregation models at sub-system level, using a technique proposed in Section 5.4.1.

4. **Import sub-system component or system level models:** The models associated with each of the logical component and system level are assumed available. These models are imported in this step.

5. **Construct a sub-system workflow:** This step, given the models imported in step-4 and generated in step-3, constructs a workflow for the sub-system under consideration, using the algorithm described in Section 5.4.2. The obtained workflow is stored for further use.

6. **Generate aggregation models at system level:** Once all the sub-systems are covered, this step generates aggregation models at system level using the method discussed in Section 5.4.1.

7. **Construct a system level workflow:** This step constructs the system level workflow, using the algorithm described in Section 5.4.2. The workflow is stored for further use.

END

5.4.1 Dynamic Generation of Models

As stated in Chapter 4, the logical view describes the system components and their inter-relations. A component in the logical view has ports assigned to it, through which, it is connected to other components. The port types are classified according to the physical entities "flowing" through the connections. A connection can only exist between the same types of the ports. The connections also define the direction of the physical flow among the components. Furthermore, a system architecture in the logical domain also contains a description/view of the hierarchical decomposition of the system.

In this section, utilising the above logical view information, a method is proposed for generating aggregation computational models. Aggregation of variables is required at two places, i.e. within a same hierarchical level and between the levels. Accordingly, two separate techniques are developed to fulfil the corresponding needs. Both types of models are employed for constructing the sizing workflows. Here, the variable aggregation involves finding the value of a variable as a function of other variables.

Within the same levels, inside the individual sub-systems (i.e. at the component level) and between the sub-systems (i.e. at the sub-system level), there are logical connections for which computational models are required to be generated. In this first technique, the aggregation models of the logical connections between the components/sub-systems of the same level are created. These models are automatically generated whenever the connections of the logical view are modified. In the second technique, the variables at one level are aggregated at the level above it. For instance, a variable at the system level may be function of variables below it. The technique only considers aggregation of additive variables, e.g. the system weight is an addition of sub-systems weights.

5.4.1.1 Generation of Logical Connection Models

There are three types of logical connection arrangements possible, as shown in Figure 5.11. Type-I represents a one-to-one connection, where there is only one source-port and one sink-port. A source-port belongs to a source-component, while, a sink-port belongs to a sink-component. A component is designated as a source or a sink depending

on the direction of the physical flow, i.e., by the arrow direction of the logical connection. Type-II (Merge) connection denotes many source-ports, but only one sink-port, while type-III (Fork) denotes the opposite, one source-port and many sink-ports.

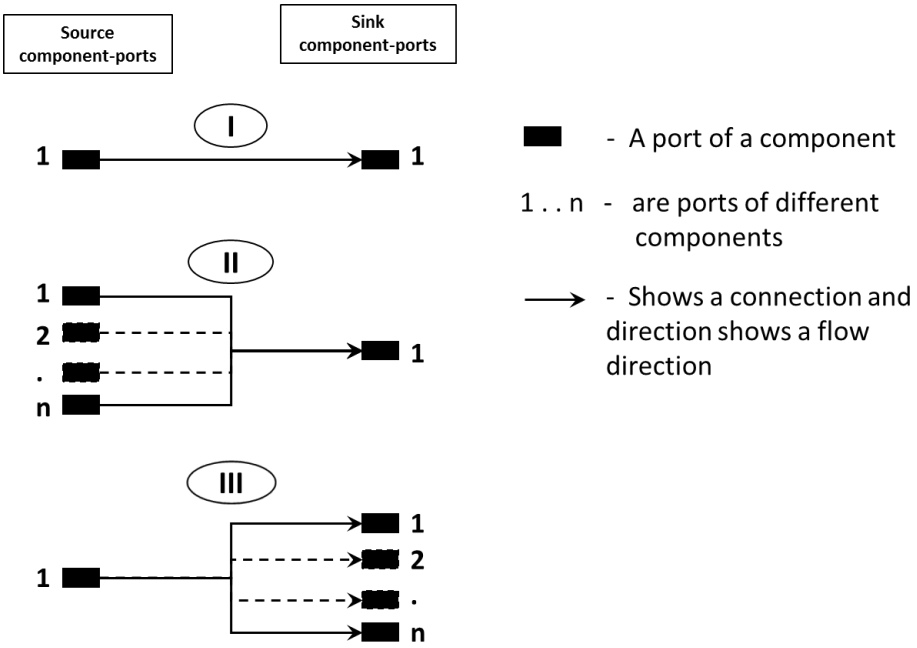


Figure 5.11: *Types of logical connection arrangements.*

There are three types of ports: material, energy and signal. Thus, in total, there are nine possible types of connections in the logical view. Depending on these combinations, connection models which utilize the port-parameters are developed . A simple example of three arrangement types for material (mass) flow ports, along with connection models for the three connection arrangements is shown in Figure 5.12. The connection models are created dynamically whenever architectural changes are made. However, it should be noted that if the user wants to give different connection models than considered in this approach, the corresponding connection models are required to be produced manually.

5.4.1.2 Generation of Aggregation Models Between the Levels

Here, only additive variables are aggregated from a level to the level above it. For instance, to find the system weight, all the weights of the sub-system components are aggregated or summed up to find the system level weight. A recursive N-ary tree traversal[91] is employed to traverse the pyramid of system hierarchy (Figure 5.2). In addition,

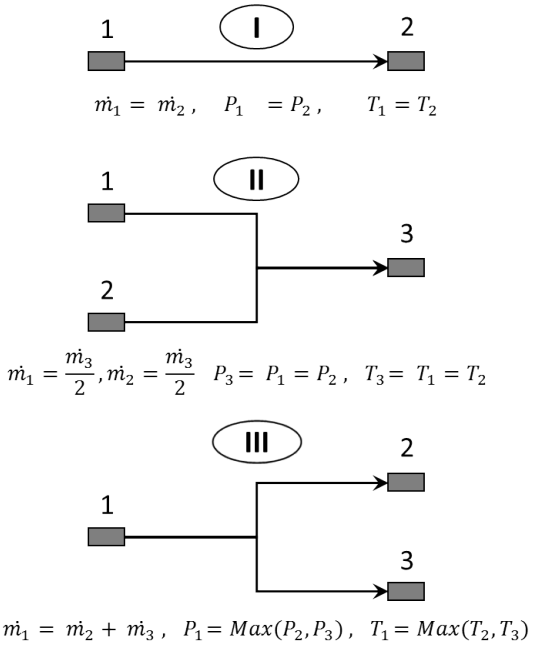


Figure 5.12: *Example of connection models for a material flow.*

the fundamental dimensions[92] are used to identify the relevant (additive) variables associated with the components of the (sub) system. For instance, to aggregate sub-systems weights, the sub-systems variables that denote their weight are required to be identified in order to find the system weight. For this purpose, an attribute called 'dimension' is assigned to each variable. It is a 7-dimensional array storing powers of the fundamental dimensions as shown in Figure 5.13. The fundamental dimensions along with their unit in International System of Units (SI) are as follow: mass (kilogram (kg)), length (meter (m)), time (second (s)), temperature (kelvin (K)), current (ampere (A)), luminous intensity (candela (cd)) and amount of substance (mole (mol)). For instance, a mass variable is represented with the fundamental dimension of mass, 'M' as '1' and the other dimensions are zero, i.e., [1, 0, 0, 0, 0, 0, 0]. Similarly, the power variable is represented as [1, 2, -3, 0, 0, 0, 0]. To aggregate particular variables during the N-ary traversal, the variables that are required to be aggregated are automatically identified using the 'dimension' attribute.

All (additive) variables required at aircraft level are aggregated to find system-level performance. A similar approach can be used to aggregate variables from components to sub-systems level.

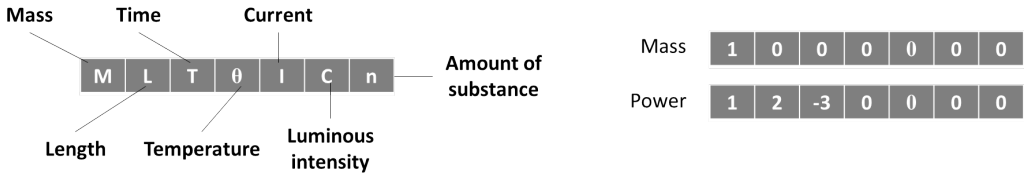


Figure 5.13: *Fundamental dimensions concept. Left - the generic array of fundamental dimensions, Right - examples for 'Mass' and 'Power'.*

5.4.2 Workflow Construction

Here, the computational sizing workflow is modelled as a bipartite graph (BG). It is assumed that the computational (behaviour) models associated with their corresponding sub-system components are available, whereas the logical connection models are generated dynamically as described above. During the sizing process, some of the variables of these models are assumed or designated as known by the user, and the rest are assumed unknown. In addition, the variables are designated as known when their values are available from the requirements placed on the system or target values set by the sink sub-system.

To decide which unknown variable should be calculated from which model, the constraint management method[93] developed to solve a system of algebraic equations having single output is extended in this research to handle computational models with multiple-inputs and outputs.

The workflow (W) is represented by the directed bipartite graph (G):

$$W = G(M, V, E) \tag{5.14}$$

where, M, V and E are sets of models, variables and edges, respectively. The following holds regarding W:

$$M \cap V = \emptyset \tag{5.15}$$

where, $\emptyset$ is a null set. In addition, for an edge, e, described by its end nodes, a and b, if $e = (a, b) \wedge e \in E$ then,

$$(a \in M \wedge b \in V) \oplus (b \in M \wedge a \in M) \tag{5.16}$$

Here, $\oplus$ - is an 'exclusive or' operator.

The flowchart describing the workflow construction process is shown in Figure 5.14. The flowchart steps are briefly described here with an example. However, the significant individual steps of the process are explained in later parts of this section.

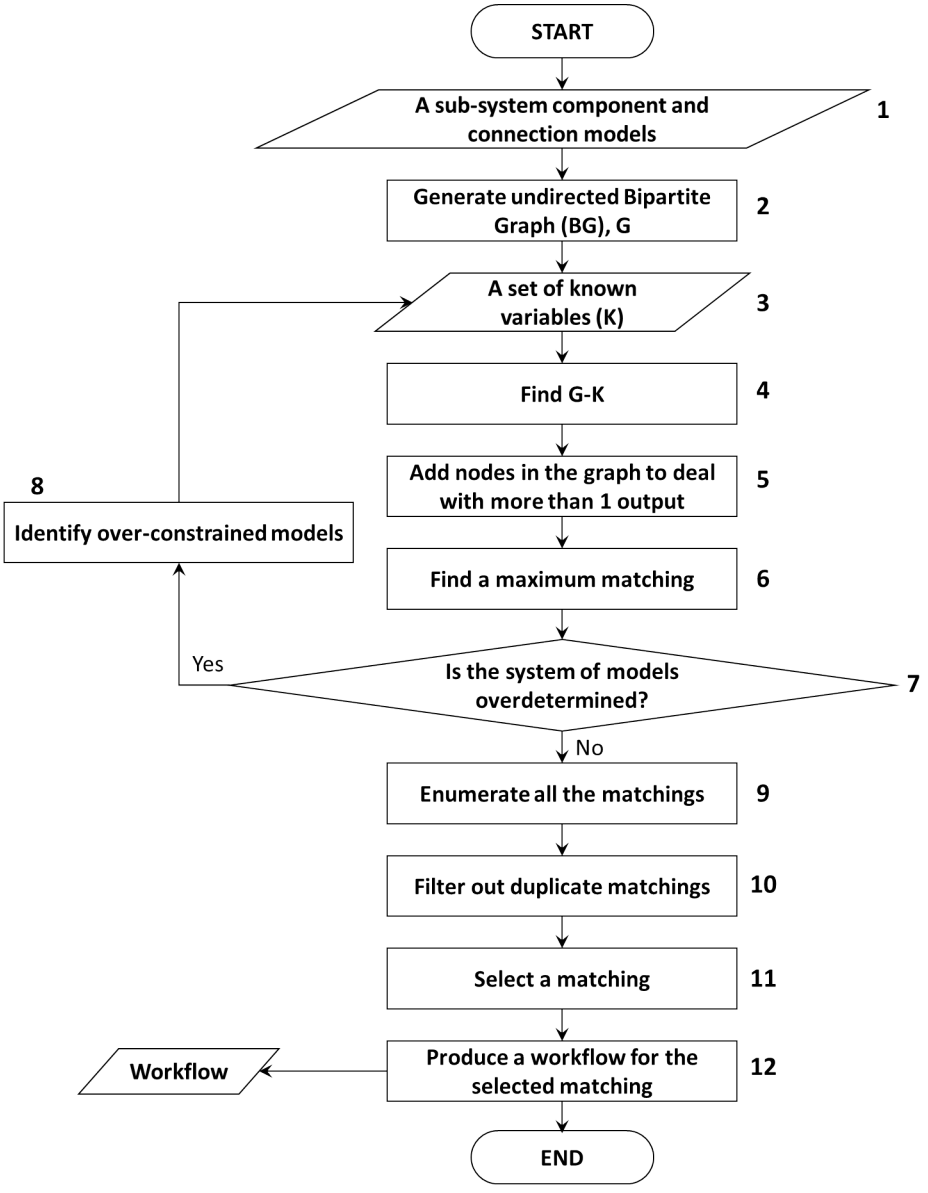


Figure 5.14: *Construction of a workflow.*

START

1. **A sub-system components and connection models:** Initially, all the sub-system

models are imported. This involves collecting all the models behind the components and connection models which are generated automatically using the methods described in the previous subsection.

2. **Generate undirected bipartite graph:** This step generates an undirected bipartite graph, $G(M, V, E)$, which describes the set of models associated with the sub-system. Here, M, V and E are model, variable and edge sets, respectively. An edge in the graph denotes association between variable and model.

3. **A set of known variables ($\mathbb{K}$):** The user provides the set of variables for which values are known. These variables are denoted by a set, $\mathbb{K}$.

4. **Find $G - \mathbb{K}$:** This step partitions the variable set, V, into two disjoint sub-sets as $\mathbb{U}$ and $\mathbb{K}$, called unknown and known variable sets, respectively, such that,

$$(\mathbb{U} \subset V \wedge \mathbb{K} \subset V) \wedge (\mathbb{U} \cup \mathbb{K} = V) \wedge (\mathbb{U} \cap \mathbb{K} = \emptyset) \tag{5.17}$$

   Once variable set (V) is partitioned into sets $\mathbb{U}$ and $\mathbb{K}$, next, undirected graph (UDG), i.e. $G - \mathbb{K}$, is created deleting known variables nodes and their associated edges, that is,

$$UDG = G(M, \mathbb{U}, UE) \tag{5.18}$$

   Here, $UE \subset E$.

5. **Add duplicate nodes:** To deal with the models with multiple outputs, duplicate model nodes are added in the UDG, along with their corresponding edges. If a model has 'n' number of outputs then the duplicate model nodes added for this model will be 'n-1'. This new graph is denoted by UDG_d.

6. **Find a maximum matching:** This step applies a maximum matching enumeration algorithm on the graph obtained in the previous step, i.e. UDG_d, to find which parameter should be calculated from which model. In this step, only one matching is found to check the system determinacy in the next step.

7. **Condition - is the system of models is overdetermined?:** This step identifies whether the system of models is over-determined. This is done by investigating

the matching. If there are any unmatched variables in the graph (see details later in this section on matching and Appendix E), it implies that there could be some over-determined models in it. If the system of models is over-determined proceed to next step, otherwise proceed to step-9.

8. **Identify over-determined models:** This step recognises the over-determined models. Detection of such models is particularly useful to identify the variables that (over) constrain the models; so that, a new set of known parameters can be proposed removing the variables from the original set of known variables that over constrain the models. Then, steps, step-3 to step-7, are again performed.

9. **Enumerate all the matchings:** Provided the system of models is not over-determined, this step enumerates all the possible maximum matchings of the UDG_d, using the maximum matching enumeration algorithm described later in this section.

10. **Filter out duplicate matchings:** Out of the enumerated matchings list, the matchings those result into duplicate workflows are removed from the list employing the algorithm proposed in Section 5.4.2.1.

11. **Select a matching:** After filtering out duplicate matchings, only one matching with the least number of reversed parameters is chosen (out of the remaining matching) to construct the workflow in the next step. The selection of a matching is performed using the proposed algorithm (see Section 5.4.2.2) which arranges the matchings such that the matching that produces a workflow with a least reversed model is at the top.

12. **Produce a workflow:** In this step, a workflow which stores the execution sequence of the involved models is produced. The details of the method employed in this step are given in Section 5.4.2.3.

END

For example, consider a set of models associated with the set of components as shown in Figure 5.15. Figure 5.15(b) shows the default undirected bipartite graph for the computational models in Figure 5.15(a).

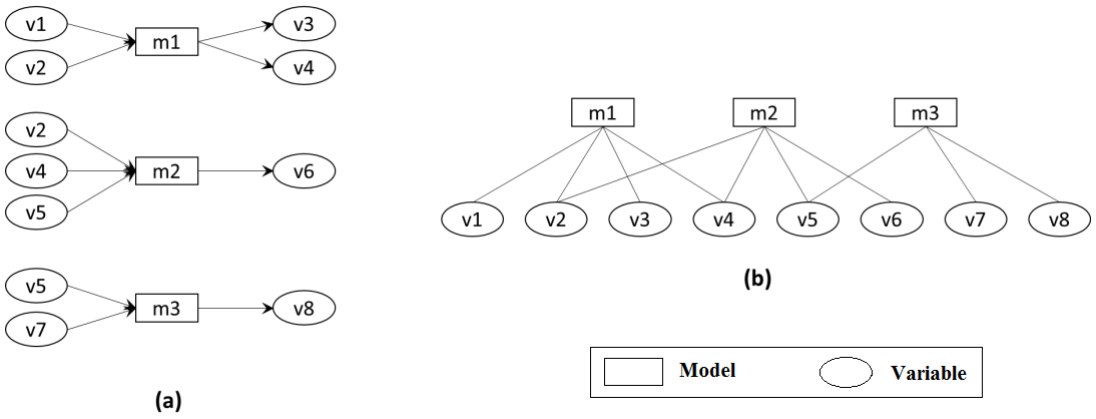


Figure 5.15: *A set of models and their undirected bipartite graph representation.*

As mentioned above, during the design process, some of the variables become known. Some of these are calculated at the sink sub-system and some are known from the requirements of the system, therefore, let's assume that some of the variables are known. These are represented by green ellipses in Figure 5.16(a). The graph is rearranged, as shown in Figure 5.16(b), with known variable nodes above and unknown variable nodes below the model nodes. Thus the partition of the variable set into known and unknown variables set is:

$$\mathbb{K} = \{v4, v6, v7, v8\} \tag{5.19}$$

$$\mathbb{U} = \{v1, v2, v3, v5\} \tag{5.20}$$

Variable set V is the union of above two sets:

$$V = \mathbb{U} \cup \mathbb{K} \tag{5.21}$$

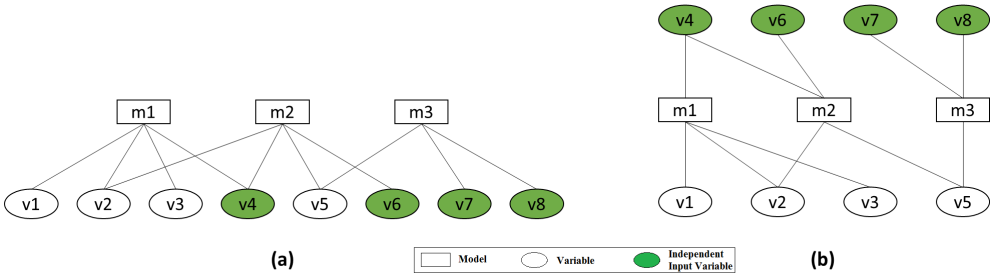


Figure 5.16: *Known (green) and unknown variables in a graph.*

To deal with a model (in this example m1) which has two output variables, a duplicate model node (m1') is added, as shown in Figure 5.17.

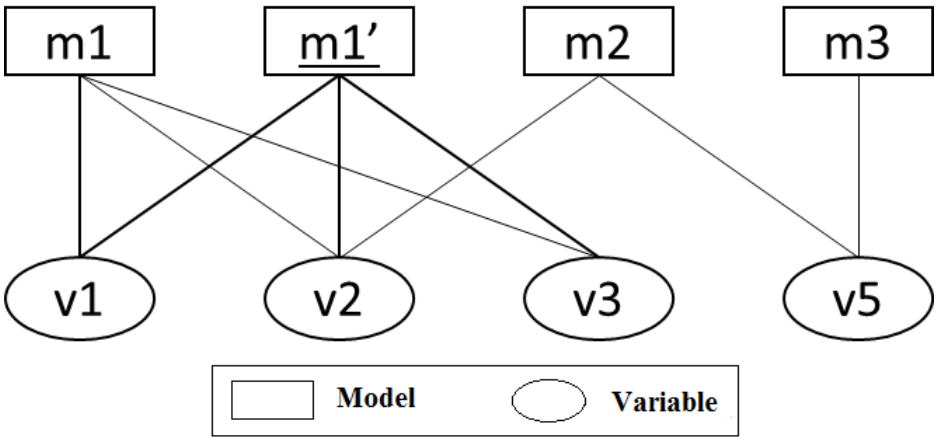


Figure 5.17: *A UDG_d with duplicate nodes.*

A maximum matching enumeration algorithm [94] is applied to the graph of Figure 5.17. The algorithm calls a recursive procedure which takes the graph as input. The procedure then divides the graph into $UDG_d^+(e)$ and $UDG_d^-(e)$ for the edge, 'e', and calls "itself" twice with each of these two new graphs as inputs, and so on. Here, the graph $UDG_d^+(e)$ is created by deleting edge e and its nodes, and the edges associated with the deleted nodes, from the original graph (Figure 5.18 - lower left); whereas, $UDG_d^-(e)$ is created by deleting only the edge from the original graph as shown in Figure 5.18 (lower right). The algorithm gives all the maximum matchings possible in the graph. Considering the graph of Figure 5.17, there are two possible maximum matchings, shown in Figure 5.19. In general, in the list of all the maximum matchings, there exist some which result in the same computational workflow. Here, both matchings result in an identical workflow, shown in Figure 5.20. Such duplicates need to be filtered out from the matchings list produced by the enumeration algorithm. From the remaining matchings, the one with the minimum reversed models is selected for constructing the sizing workflow, and the latter is solved to find the unknown variables values. To execute the workflow, firstly, a model-model DSM is extracted from the workflow. This DSM shows the dependency of a model on the other models for its inputs. Next, SCCs contained in the DSM are found, and then, the SCCs and other models of the DSM are orchestrated. To solve the SCCs and any reversed models, mathematical (numerical) treatments such as fixed point iteration [95] [18] and optimization [18] [96] are employed.

As shown in Figure 5.19, MATCH-1 has edge set, m1-v1,(m1')-v3,m2-v2,m3-v5, and

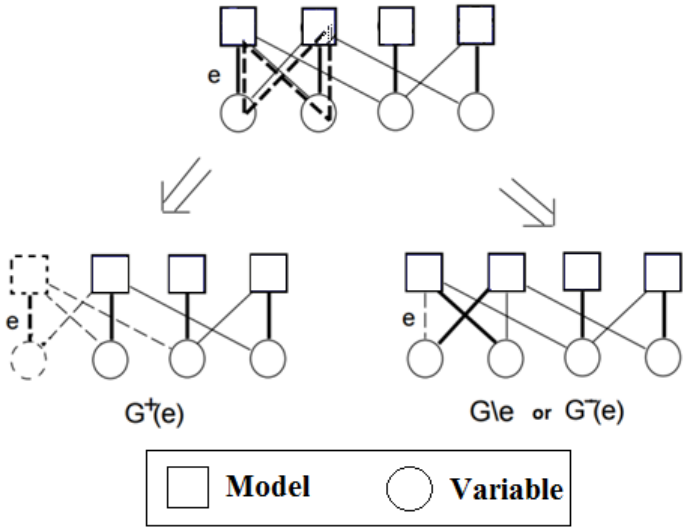


Figure 5.18: *Definition of $UDG_d(e)$ and $UDG_d(e)$ (adapted from Uno[94])*

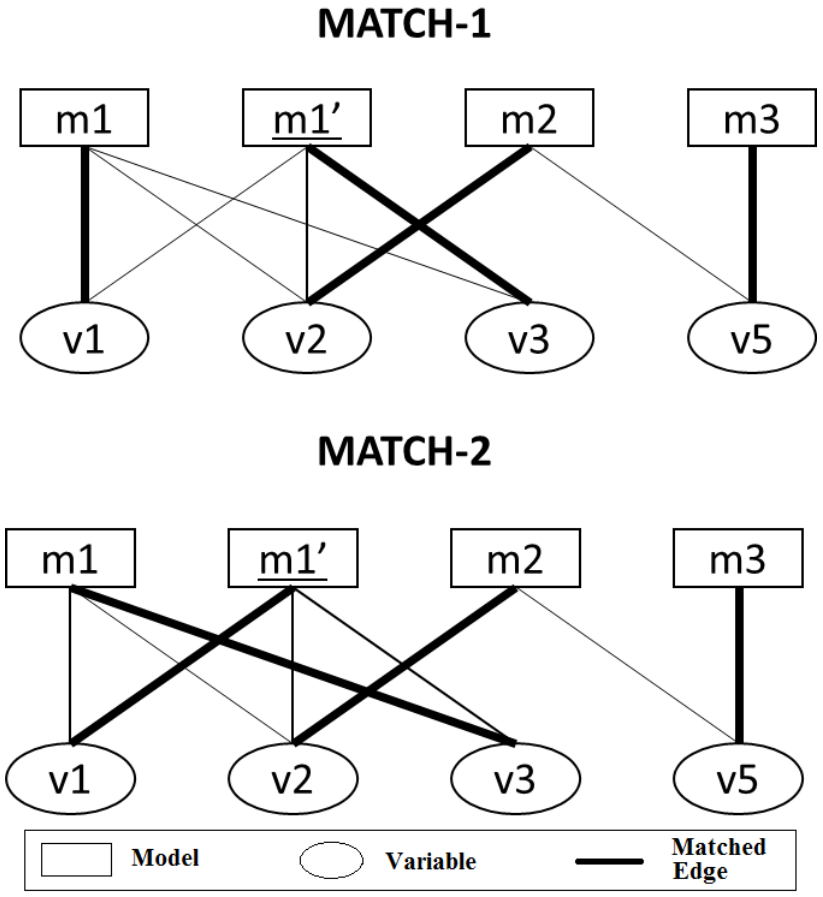


Figure 5.19: *All possible maximum matchings.*

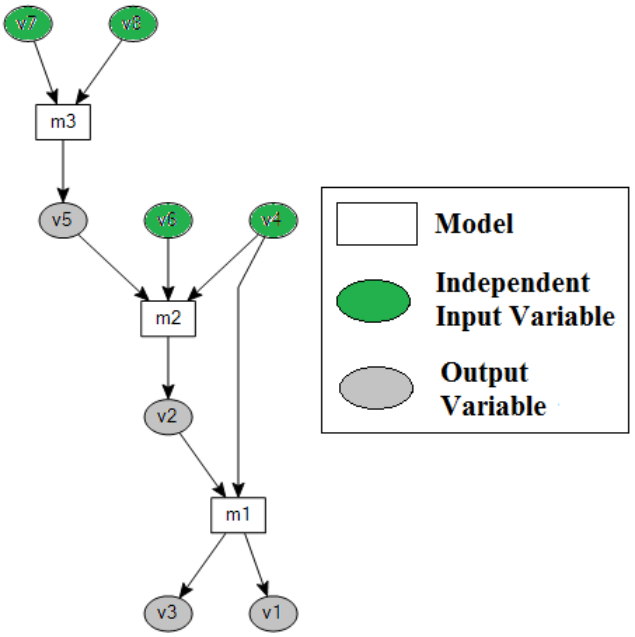


Figure 5.20: *Variable flow representation for the selected matching.*

MATCH-2 has edge set, m1-v3,(m1')-v1,m2-v2,m3-v5. Here, it can be seen that, although the algorithm produces two matchings, the resulting workflows are same. The matching edges associated to the original node m1 and its duplicate node m1' are different in the first matching and the second matching. Since both the nodes (original and duplicate) represent the same model, and the variable nodes associated with the edges corresponding to these nodes are also same. As a result, although the matching algorithm produces two different matchings, the resulting workflow is the same as shown in Figure 5.20, where only the original model node (m1) is plotted representing both matched edges to v1 and v3, from it (i.e. original and duplicate model). Here, the matchings that results into same workflows are referred to as duplicate matchings and these matchings are generated due to presence of duplicate model nodes only.

Filtering out such duplicate matchings is needed to obtain distinct workflows at the end. The next section describes the method developed to filter out such matchings.

5.4.2.1 Filtering of Duplicate Matchings

The enumeration algorithm mentioned in the previous section produces a list of possible matchings for a given UDG_d. Some of these result in identical workflows. To filter out

such matchings, distinct prime numbers (PNs) are assigned to edges of the UDG_d. For example, the assignment of PNs to edges of UDG of Figure 5.16(b) is shown in Figure 5.21. When duplicate model nodes are added, their associated edges are assigned same PNs as that of the original model node. For instance, for the UDG_d shown in Figure 5.17, the assignment of PNs to the edges of the duplicate model node, m1', is shown in Figure 5.22.

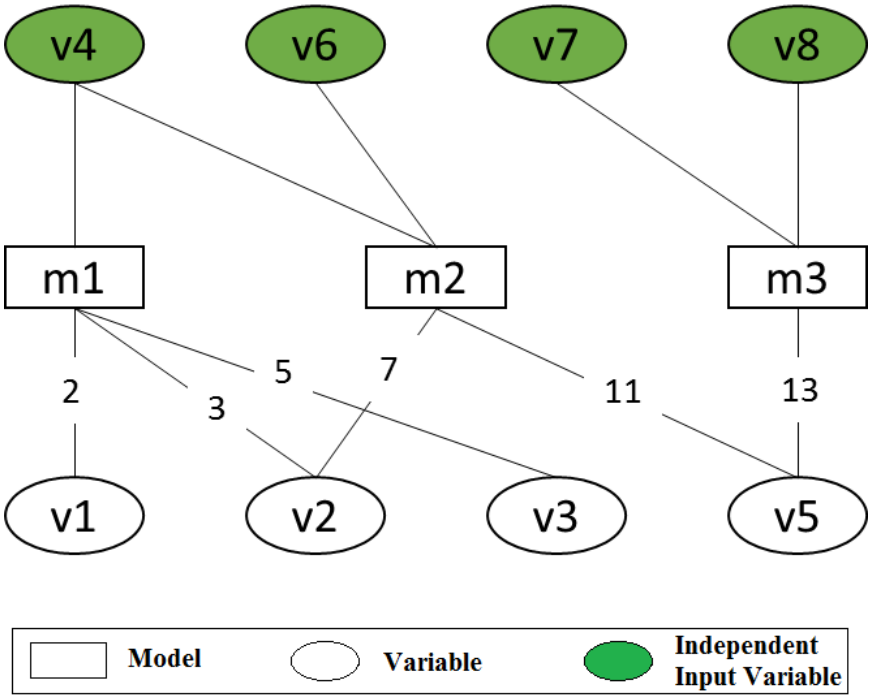


Figure 5.21: *Assignment of distinct PNs to edges.*

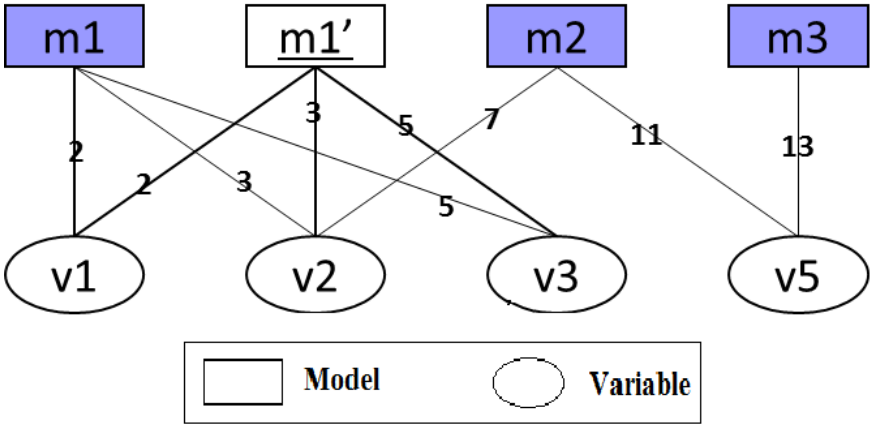


Figure 5.22: *PNs assigned to edges of a duplicate node.*

To identify the duplicate matchings, a product of the PNs assigned to edges in the matching is calculated and compared with the product obtained for other matchings.

If two matchings give the same product then they result in the identical workflows. For instance, there are two matchings possible for the UDG_d shown in Figure 5.22. These matchings are shown in Figure 5.19, and product of PNs assigned to their edges is shown in Table 5.9. Here, it can be seen that the product of the PNs for these two matchings, i.e. 910, is the same and therefore results in identical workflows as mentioned in the previous section.

TABLE 5.4: Product of PNs assigned to matching edges.

Models / Matchings	m1	m1'	m2	m3	Product
MATCH-1	V1	V3	V2	V5	910
	2	5	7	13	
MATCH-1	V3	V1	V2	V5	910
	5	2	7	13	

The algorithm for filtering out the duplicate matching and obtaining distinct matchings is shown in Algorithm 5.1.

The proof of the technique employed in filtering out the duplicate matchings is given below.

Statement:

"Any two matchings result in identical workflows if and only if the products of prime numbers (PNs) assigned to their matched edges are equal, otherwise the workflows are distinct."

Before proving the above conditional statement, some terms that will be used in the proof are defined.

A **predicate** is a statement that may be true or false depending on its variables (predicate variables). For instance, $P(x)$ is a predicate stating 'x is prime number'. Here, if $x = 3$ then $P(x)$ is true, and if $x = 4$ then $P(x)$ is false.

The above conditional statement, using predicates, can mathematically be written as follow.

$$P(M1, M2) \iff Q(w1, w2) \textbf{ OR } \neg P(M1, M2) \iff \neg Q(w1, w2) \tag{5.22}$$

Algorithm 5.1: Filtering out of duplicate matchings.

Input : Undirected graph, UDG obtained after deleting known variables nodes and their associated edges i.e. $G(M, \mathbb{U}, UE)$, undirected graph with duplicate nodes obtained by adding duplicate nodes of models having more than one output and their associated edges(i.e. UDG_d), list of matchings obtained for the (graph) UDG_d, $L_{Matchings}$ and list of prime numbers, L_{PNs}

Output: A list of distinct matchings, $L_{Filtered}$

```
integer k = 0;
for edge edg in UE do
    Assign k^th prime number of L_PNs to edg;
    k = k + 1 ;
Assign the prime numbers assigned to edges of UDG to corresponding edges in UDG_d
;   // In case of duplicate nodes, assign prime numbers to edges same as that of
the original node edges
for Matching M in L_Matchings do
    integer product = 1;
    for Edge edg in M do
        product = product × Prime number assigned to edge, edg ;
    Associate product to M;
for Matching M in L_Matchings do
    if L_Filtered does not contains matching with product equal to product associated to
    M then
        Add M to L_Filtered;
return L_Filtered ;
```

where,

$P(M1, M2)$: Products of PNs assigned to sets of edges, $M1$ and $M2$, are same.

$Q(w1, w2)$: Workflows, $w1$ and $w2$ are same.

The proposition $P(M1, M2) \iff Q(w1, w2)$ is equivalent to,

$$[P(M1, M2) \implies Q(w1, w2)] \wedge [Q(M1, M2) \implies P(w1, w2)] \tag{5.23}$$

In order to prove the double implication statement ($P(M1, M2) \iff Q(w1, w2)$), both $P(M1, M2) \implies Q(w1, w2)$ and $Q(M1, M2) \implies P(w1, w2)$ are required to be proved.

Definitions and assumptions:

1. For a graph, $G = (V_m, V_v, E)$, a matching, M , of G is a subset of the edges, E , such that no vertex in $V(= V_m \cup V_v)$ is incident to more than one edge in M.

2. Duplicate matchings exist only when one or more of the models have more than one output. The number of duplicate model nodes added for such a model is $(n-1)$, where n is a number of outputs of the model.

3. Models having exactly one output do not require duplicate nodes to be added, and these models do not contribute in producing duplicate matchings; therefore, matching edges associated with these models in $M1$ and $M2$ are same.

4. All the nodes of the graph, G, are matched (i.e. at most one edge of the matching is incident on the nodes).

5. The Fundamental Theorem of Arithmetic [97] postulates that every integer greater than one, N can be expressed as the product of powers of prime numbers:

$$N = p^{\alpha} \times q^{\beta} \times ... \times r^{\gamma}$$

Where, $p, q, , r$ are prime, while $\alpha, \beta, ..., \gamma$ are positive integers. Such representation is unique up to the order (power) of the prime factors

6. PNs assigned to edges of G are distinct except duplicate nodes. For duplicate nodes, the associated edges are assigned same PNs as that of corresponding edges' PNs of the original node.

Proof.

- Let 'a' be the number of models that have more than one outputs, and $x_1, x_2, ..., x_a$ are the numbers of outputs from a models. Therefore, the total number of duplicate nodes in the graph G is:

$$x = \sum_{i=1}^{a} (x_i - 1) = (x_1 - 1) + (x_2 - 1) + ... + (x_a - 1) \tag{5.24}$$

- Thus, total number of models, μ

$$\mu = |V_m| - x$$

- The cardinality of any maximum matching on the graph, G is ψ, and $\psi = |V_m|$

- Suppose product of PNs of the edges in $M1$ and $M2$ are given by π_1 and π_2, respectively. The products, each containing ψ number of PNs, can be written as follow.

$$\begin{aligned}\pi_1 = (\rho_1 \times \rho_{d1,1} \times \rho_{d1,2} \times ... \rho_{d1,x_1-1}) \\ \times (\rho_2 \times \rho_{d2,1} \times \rho_{d2,2} \times ... \times \rho_{d2,x_2-1}) \times ... \\ \times (\rho_a \times \rho_{da,1} \times \rho_{da,2} \times ... \times \rho_{da,x_a-1}) \\ \times \boldsymbol{\rho_{(a+1)}} \times ... \times \boldsymbol{\rho_{(\mu-1)}} \times \boldsymbol{\rho_\mu}\end{aligned} \tag{5.25}$$

$$\begin{aligned}\pi_2 = (\sigma_1 \times \sigma_{d1,1} \times \sigma_{d1,2} \times ... \sigma_{d1,x_1-1}) \\ \times (\sigma_2 \times \sigma_{d2,1} \times \sigma_{d2,2} \times ... \times \sigma_{d2,x_2-1}) \times ... \\ \times (\sigma_a \times \sigma_{da,1} \times \sigma_{da,2} \times ... \times \sigma_{da,x_a-1}) \\ \times \boldsymbol{\sigma_{(a+1)}} \times ... \times \boldsymbol{\sigma_{(\mu-1)}} \times \boldsymbol{\sigma_\mu}\end{aligned} \tag{5.26}$$

  Where, ρ and σ are PNs. The PNs in the products are grouped as per PN(s) of associated edge(s) from each model(s) (original model and its corresponding added models in case of a model having more than one output). These groups are represented by showing the PN groups in brackets. For example, the underlined bracket in the product equations represents the product of the PNs associated to edges from an original model and its duplicates. Here (Equation 5.25), ρ_1 is associated to original model node, whereas, $\rho_{d1,1} \times \rho_{d1,2} \times ... \rho_{d1,x_1-1}$ are the PNs associated to the respective duplicates. The PNs which are without brackets are associated with the matching edges from models having single output. Such PNs are shown in bold in Equation 5.25 and 5.26.

- The number of models which have exactly one output is $(\mu - a)$, and the edges associated with these models do not contribute to producing duplicate matchings. If the matched edges incident on these models are same in both the matching, then the PNs associated to those edges (shown in bold) are considered to be exactly

the same in the above products because in both the matchings ($M1$ and $M2$), the variables incident on the edges are same and, therefore, the associated PNs are same.

- In products, Equation 5.25 and 5.26, an individual bracket represents the product of PNs associated to edges of models with more than one output and to its duplicate model nodes. If the products, π_1 and π_2 are to be the same, their remaining PNs of the products should be the same (as postulated by the number theorem). However, this is possible only if all the brackets in first product contain the same PNs as that of the corresponding bracket in the second product. This means that the original model node, considering all matching variables of its duplicate model node, is matched to the same variables in both the matchings. The edges associated to these variables have been assigned distinct PNs which are the same in the corresponding brackets in both the product equations. Thus, for the model, all the output variables in both resulting workflows are same. This is valid for other brackets as well.

- This concludes that, if the products of the PNs of the edges of any two matchings are same then the resulting workflows are also same.

$$\therefore P(M1, M2) \implies Q(w1, w2) \tag{5.27}$$

The alternative statement, $Q(w1, w2) \implies P(M1, M2)$ can also be proved in a similar fashion.

$$\therefore P(M1, M2) \iff Q(w1, w2)$$

■

5.4.2.2 Selection of a Matching

After filtering out the duplicate matchings as described in the previous section, it is possible that there still exists more than one matching which produce distinct workflows.

However, only one matching is required to be selected, so that a corresponding workflow is produced. In this section, a proposed method, to rank the filtered matchings, is described. The matchings are ordered in descending order of the number of reversed variables in their corresponding workflows. That is, the matching producing a workflow with the least number of reversed variables is selected.

As mentioned in Chapter 2, the reversed/modified models require numerical treatment to solve them, and therefore, such models require the extra computational cost to solve them as compared to their corresponding default models. It is assumed that the computational cost of reversing any variable of the models is same, and therefore, a workflow containing less number of reversed variables is computationally efficient. However, the reversal cost is different for different variables and usually dependent on the model (i.e. equations behind it) and the variable to be reversed. During selecting a workflow, it is important to consider the individual reversal costs associated with the reversed variables in a workflow. It is not within the research scope to determine the computational cost of reversing the individual (reversed) variables of the workflow. However, the method is proposed that can take into consideration different reversal cost of these variables, provided the normalised computational (reversing) costs of the variables are available.

To order the list of matchings, a candidate variable to be reversed as an output, is assigned with a reversal weight in the model's default condition. For the models in Figure 5.15, the assignment of weights to the variables is shown in Figure 5.23. For demonstration purpose, unit weights are assigned to the input, whereas, the output variable weights are zero; however, it is possible to assign different weight to the output variables depending on the involved computational cost in reversing corresponding output variable, provided the computational cost of reversing output variables is available. It can be seen here that model's input variables are assigned higher weight than its output variables. The assignment is performed in this way with the following consideration in mind: 1) if a variable that is an output from a model in its default condition and is also output (from the model) in the workflow then the computational cost associated with this variable is low (therefore, assigned weight is 0), 2) if a variable is input to a model in its default condition and is output (from the model) in the workflow then the

computational cost associated with this variable is higher (therefore, assigned weight is 1).

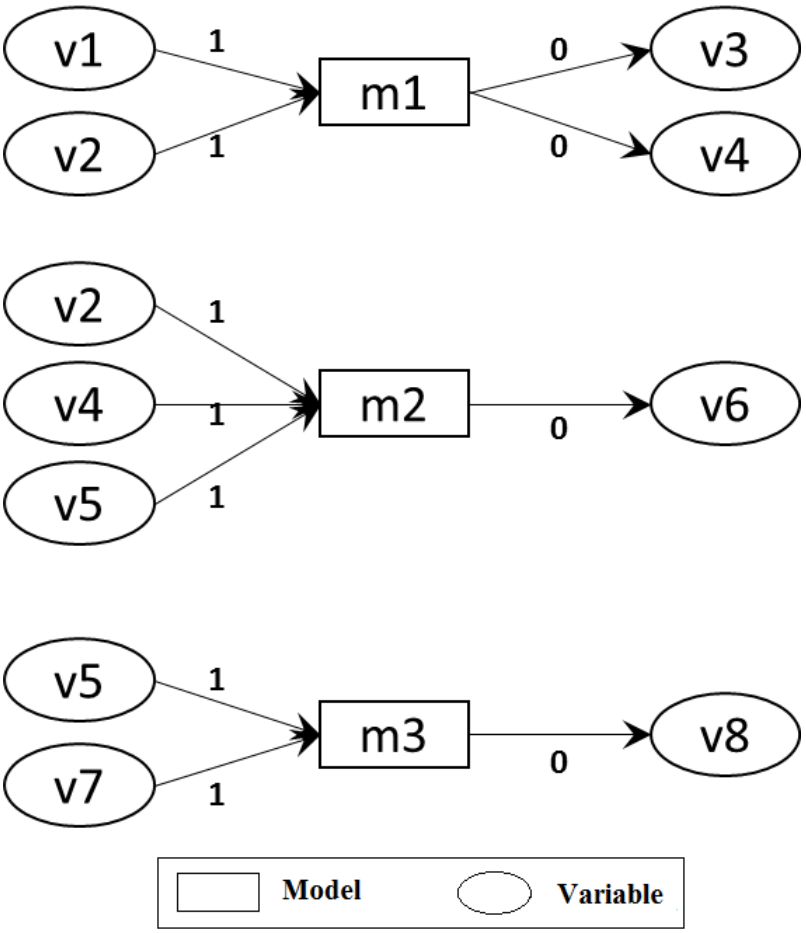


Figure 5.23: *Example of reversal weight assignment to the variables in their default condition.*

Algorithm 5.2 takes a list of distinct matchings as input and gives out ordered list of matching.

The matchings obtained for the set of models of Figure 5.23 are shown in Figure 5.19. For these matchings, the sum of reversal weights is shown in Table 5.5. For these matchings, the sum of the weights is same, and therefore, any matching out of these matchings can be selected to construct the workflow.

Algorithm 5.2: Ordering of matchings.

Input : A list of distinct matchings ($L_{Filtered}$)
Output: Sorted list of matchings (L_{sorted})

```
Assign reversal weights to variables;
for Matching M in L_Matchings do
    integer sum = 0;
    for Edge edg in M do
        sum = sum + weight assigned to variable incident on the edge, edg ;
    Associate sum to M;
L_Sorted =Sort the list L_Matchings in descending order of their associated sum;
return L_Sorted ;
```

Table 5.5: Sum of the revrsal weights assigned to the matched variables

Matchings \ Models	m1	m1'	m2	m3	Sum
MATCH-1	V1	V3	V2	V5	3
	1	0	1	1	
MATCH-1	V3	V1	V2	V5	3
	0	1	1	1	

5.4.2.3 Producing a Workflow

As described in the flowchart shown in Figure 5.14, once a single matching is selected out of the available options, next, the workflow corresponding to the matching is produced. For this, the method proposed by Balachandran[18] is adopted here. This section outlines the process of production of a workflow from the selected matching.

As Balachandran's method requires an incidence matrix as the input, first, from the selected matching, an incidence matrix, storing information about the input to and output from the models, is generated. Next, a model to model design structure matrix (DSM) is created from the incidence matrix. The DSM stores models interdependency (i.e. an output from a model is required as an input to another model). Finally, rows (models) of the DSM are sequenced to produce the workflow (i.e. models' execution sequence).

The method is demonstrated for the matching in Figure 5.19 and corresponding variable flow representation (i.e. incidence matrix graph) in Figure 5.20. The incidence matrix corresponding to the matching is shown in Table 5.6. Here, 'i' and 'o' in the DSM stand

for inputs and outputs, respectively. For example, for the element in first row (m1) and the second column (v2), 'i', denotes that variable, v2 is an input to model, m1.

TABLE 5.6: Icidence matrix

	v1	**v2**	**v3**	**v4**	**v5**	**v6**	**v7**	**v8**
m1	o	i	o	i				
m2		o		i	i	i		
m3					o		i	i

The model DSM, for this incidence matrix, is shown in Table 5.7.

TABLE 5.7: Model DSM

	m1	**m2**	**m3**
m1	1	1	0
m2	0	1	1
m3	0	0	1

In order to sequence the models, first, any coupled models (i.e. strongly connected component (SCC)) of the matrix are found[90]. Next, a new matrix is generated representing the SCCs as a single entry in the matrix. Then the matrix rows of the new matrix are sequenced[89] to obtain execution sequence of models/SCCs.

There is no SCC in the above DSM, see Table 5.7. Table 5.8 shows the sequenced DSM.

TABLE 5.8: Sequenced DSM

	m3	**m2**	**m1**
m3	1	0	0
m2	1	1	0
m1	0	1	1

The execution sequence of the models obtained from the sequenced DSM (shown above), is m3 → m2 → m1. If there were any SCC, the SCC would have been represented in the sequence as a single aggregate entry. However, there is a need to sequence the models of the SCC to reduce the computational cost. The workflow object produced at the end stores the sequence of non-coupled models and (coupled models) SCCs if there are any, as well as the sequence of the models inside the SCCs.

The next part describes the sequencing of the models of the SCC employed in the presented research.

It is not possible to sequence the models of a SCC using the approach described above for sequencing non-coupled models. In the case of non-coupled models, it is possible to obtain a lower triangular matrix which is a sequenced DSM. However, in the case of coupled models (SCCs), it is not possible to get a lower triangular matrix and so the SCCs are represented as a single entry in the matrix. An algorithm used for drawing a directed graph is employed to sequence the SCC models. Firstly, the directed graph representing the variable flow between the models of a SCC is created, and then, the directed graph drawing algorithm proposed by Sugiyama[98] is employed to arrange the model nodes in layers such that there exists a minimum crossing of the edges between the models. The output of the algorithm is an arrangement of nodes (models) to draw the graph with minimum crossing of graph-edges. The arrangement is described by the layers and columns to locate the nodes of the graph. The layers are arranged such that there are only directed links to the nodes from nodes in the previous layers. In order to extract the sequence of models (nodes), the layer information output from the algorithm is utilised. The (SCC) models sequence is produced by traversing the layers (in a given arrangement) and adding models (nodes) of the layer in the sequence. An illustrative example is given in Appendix E (for more details on Sugiyama algorithm see reference[98]).

5.4.3 Determinacy

Depending on the available known variables (designated as known by the user), a system of models could be determined, under-determined or over-determined. The system of models is under-determined if the number of known variables is smaller than the required number by the system to compute all unknown variables. In the case of an over-determined system, the number of known variables is greater than the required number for solving the system. In case of a determined system of models, the number of user-specified known variables is exactly same as that of the number required for solving. Balachandran[18] proposed the following criteria for determining the solvability of the system of models as shown in Equation 5.28, 5.29 and 5.30. These criteria are only a necessary condition for the system to be determined, but not the sufficient

condition. That is, even though one of these criteria is satisfied one of the models of the system could still be over-determined.

$$TNvar - NIvar - Noutmod = 0 \quad \text{Determined} \tag{5.28}$$

$$TNvar - NIvar - Noutmod > 0 \quad \text{Under-determined} \tag{5.29}$$

$$TNvar - NIvar - Noutmod < 0 \quad \text{Over-determined} \tag{5.30}$$

Where, $TNvar$ - Total number of variables
$NIvar$ - Number of independent variables (i.e. known variables)
$Noutmod$ - Sum of the total number of outputs of each model in a system
Accordingly, the process of creating the workflow is demonstrated on the set of models shown in Figure 5.15(a) for all the possible cases of the (models) system i.e. determined, under-determined and over-determined as described below.

5.4.3.1 Case-1: Determined System of Models

For demonstration purposes, let's assume that some of the parameters are known. The known parameters are shown by green ellipses in the graphs in Figure 5.24. The graph at the top in Figure 5.24 is rearranged with known variable nodes above and unknown variable nodes below the model nodes, as shown at the bottom of the figure. Thus the partition of the variable set into known($\mathbb{K}$) and unknown ($\mathbb{U}$) variable sets is:

$$\mathbb{K} = \{v2, v6, v7, v8\} \tag{5.31}$$

$$\mathbb{U} = \{v1, v3, v4, v5\} \tag{5.32}$$

Variable set V is the union of above two sets, that is,

$$V = \mathbb{U} \cup \mathbb{K} \tag{5.33}$$

Here,

$$TNvar - NIvar - Noutmode \implies 8 - 4 - 4 = 0$$

Hence, the system of models for the given known parameters is determined.

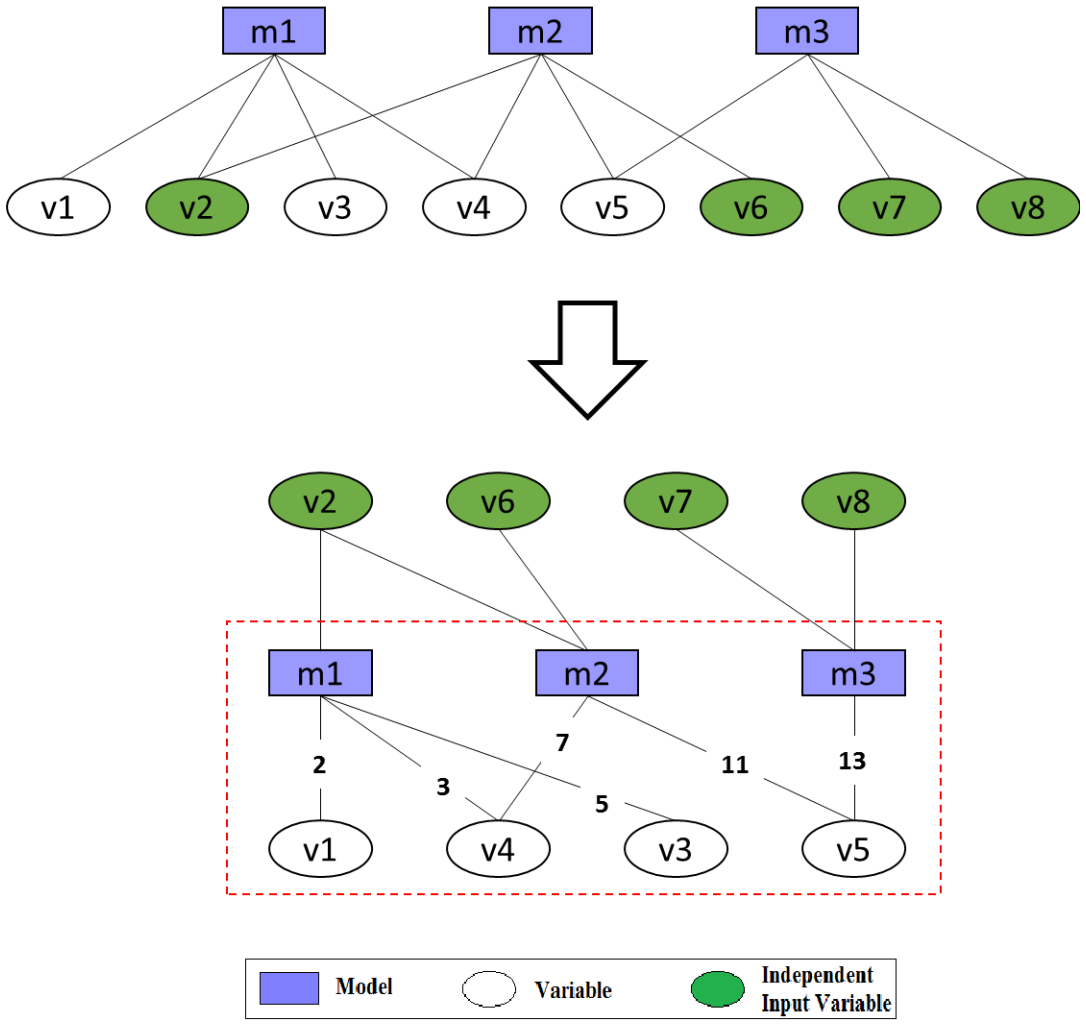


Figure 5.24: *Case-1: Known and unknown variables in a graph.*

To deal with a model with more than one output variables, duplicate model nodes are added in the UDG (marked by a red-dotted rectangle in Figure 5.24) to obtain a graph, UDG_d, as shown in Figure 5.25 (a). Here, model (m1) has two outputs, and therefore a duplicate node (m1') is added. An undirected graph (shown on the left of Figure 5.25) is given as input to the maximum matching enumeration algorithm[94] is applied on UDG_d. The algorithm gives two possible matching as shown on the right of Figure 5.25 (b). In fact, the matchings 'MATCH-1' and 'MATCH-2' results into the same workflow as shown in Figure 5.26.

In order to remove such duplicate matching, first, prime numbers are assigned to edges of the graph as shown in the graph on the left of Figure 5.25. The proposed filtering

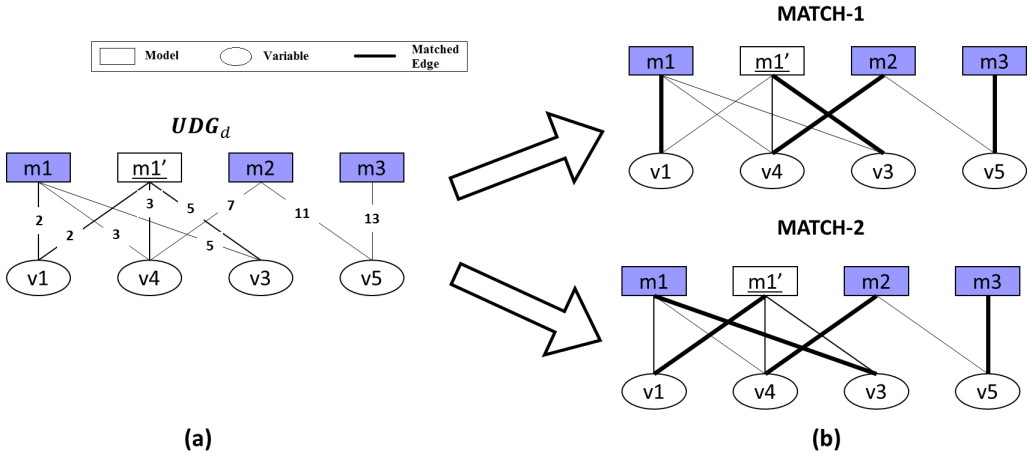


Figure 5.25: *Case-1: Enumeration of all possible matchings.*

algorithm checks if the product of prime numbers assigned to edges in a matching is the same as the product calculated for another matching. If the product is same then these two matchings result into the same workflow; hence, only one of these matching is kept. Table 5.9 shows the matchings and their product. For both the matchings, the product is the same i.e. 910. As there is only one distinct workflow (in the filtered list of matchings) possible, therefore, there is no need to perform selection algorithm on the set of filtered matching. The workflow corresponding to the matching is shown in Figure 5.26.

TABLE 5.9: Case-1: Matchings and their corresponding product of prime numbers.

Models / Matchings	m1	m1'	m2	m3	Product
MATCH-1	V1	V3	V4	V5	910
	2	5	7	13	
MATCH-1	V3	V1	V4	V5	910
	5	2	7	13	

5.4.3.2 Case-2: Under-determined System of Models

Let's assume that known parameters are v4, v6 and v8. Figure 5.27 shows a bipartite graph with known and unknown variables. The known and unknown variable sets are:

$$\mathbb{K} = \{v4, v6, v8\} \tag{5.34}$$

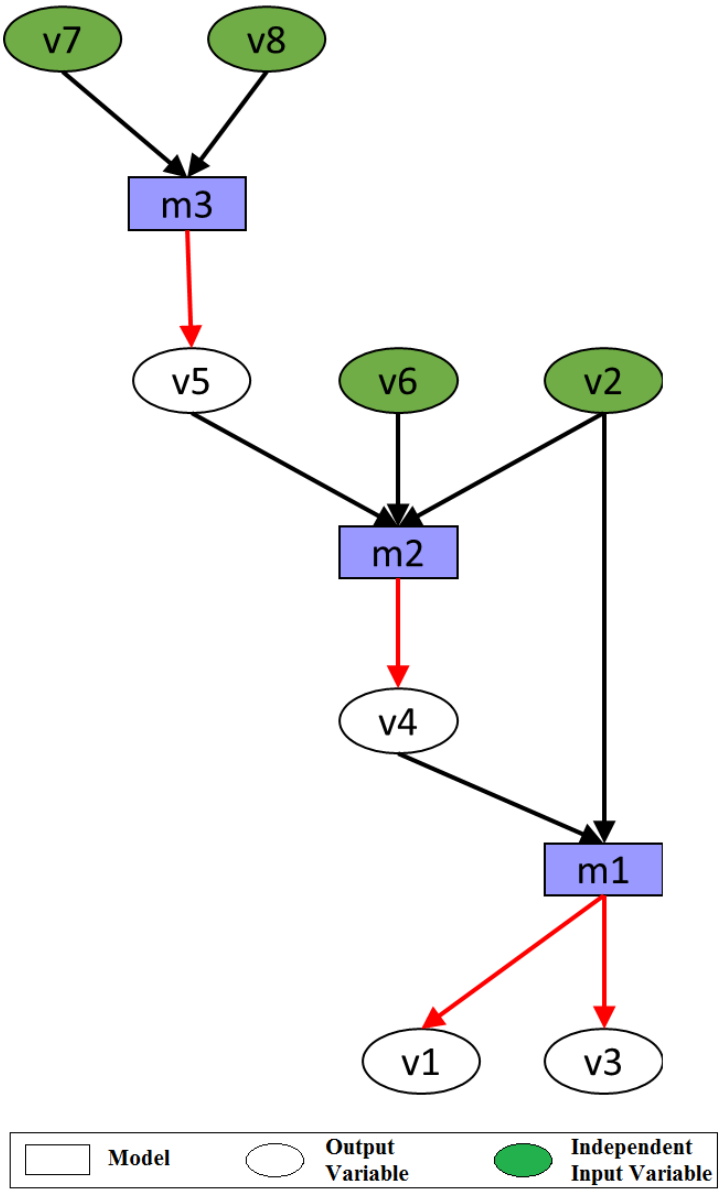


Figure 5.26: *Case-1: Workflow of the selected matching.*

$$\mathbb{U} = \{v1, v2, v3, v5, v5\} \tag{5.35}$$

Variable set V is the union of above two sets, that is,

$$V = \mathbb{U} \cup \mathbb{K} \tag{5.36}$$

Here,

$$TNvar - NIvar - Noutmode \implies 8 - 3 - 4 = 1$$

$$\therefore TNvar - NIvar - Noutmode > 0$$

Hence, the system of models for the given known parameters is under-determined.

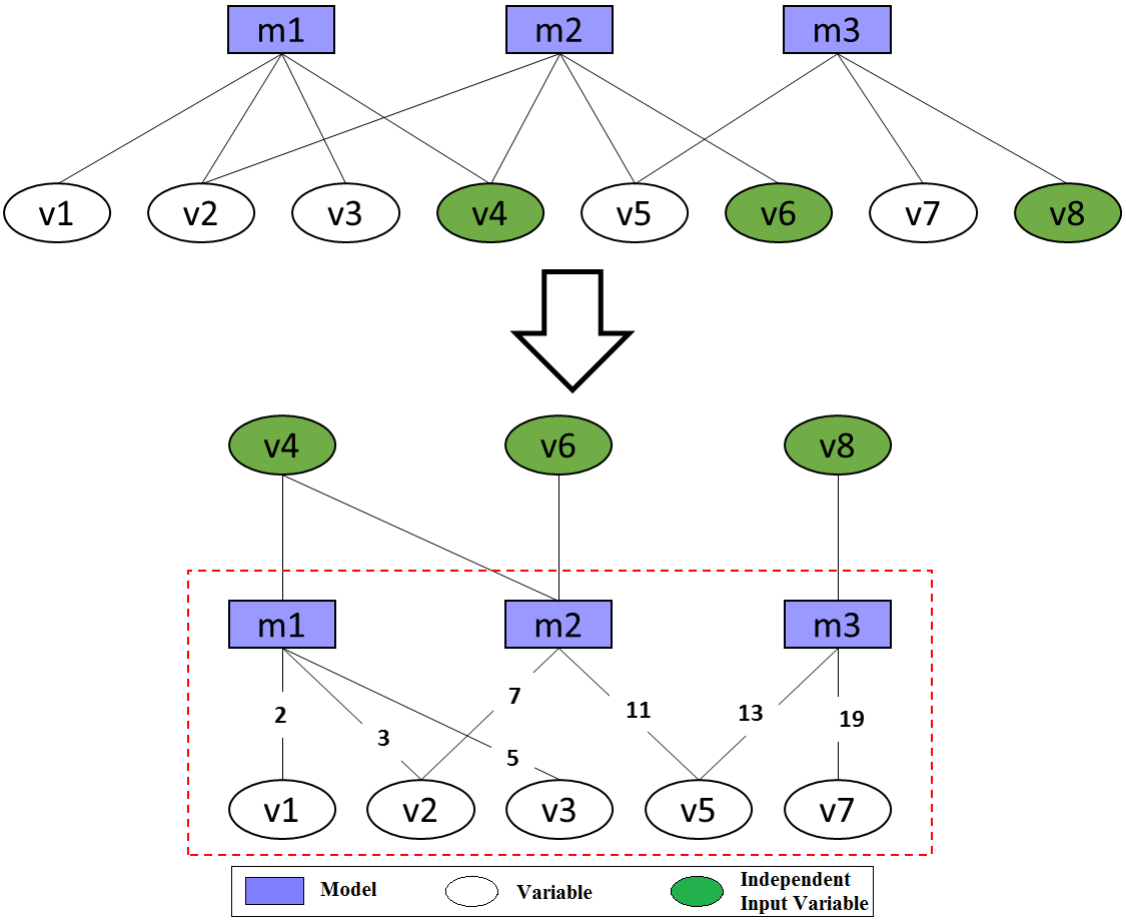


Figure 5.27: *Case-2: Known and unknown variables in a graph.*

Figure 5.27 also shows the prime numbers assigned to edges associated with unknown variables of the graph, UDG (marked by a red-dotted rectangle in Figure 5.27). The enumeration algorithm for finding all possible matchings is applied on the graph, UDG_d, which is obtained adding duplicate model nodes (i.e. m1') in UDG. Figure 5.28 shows all possible matching obtained for the UDG_d.

In Figure 5.28, there could be matchings which give identical workflows. Therefore, first, the list of matchings produced is filtered out. To do this, a product of PNs assigned to matched-edges is found out as shown in Table 5.10. Only one matching is kept out of those matchings giving the same product of PNs assigned to matched edges. The matchings after filtering out duplicates are: MATCH-1, MATCH-3, MATCH-5, MATCH-7, and MATCH-9. Out of these matchings, only one is selected to construct the workflow. These matchings are required to be ordered as per the number of reversed

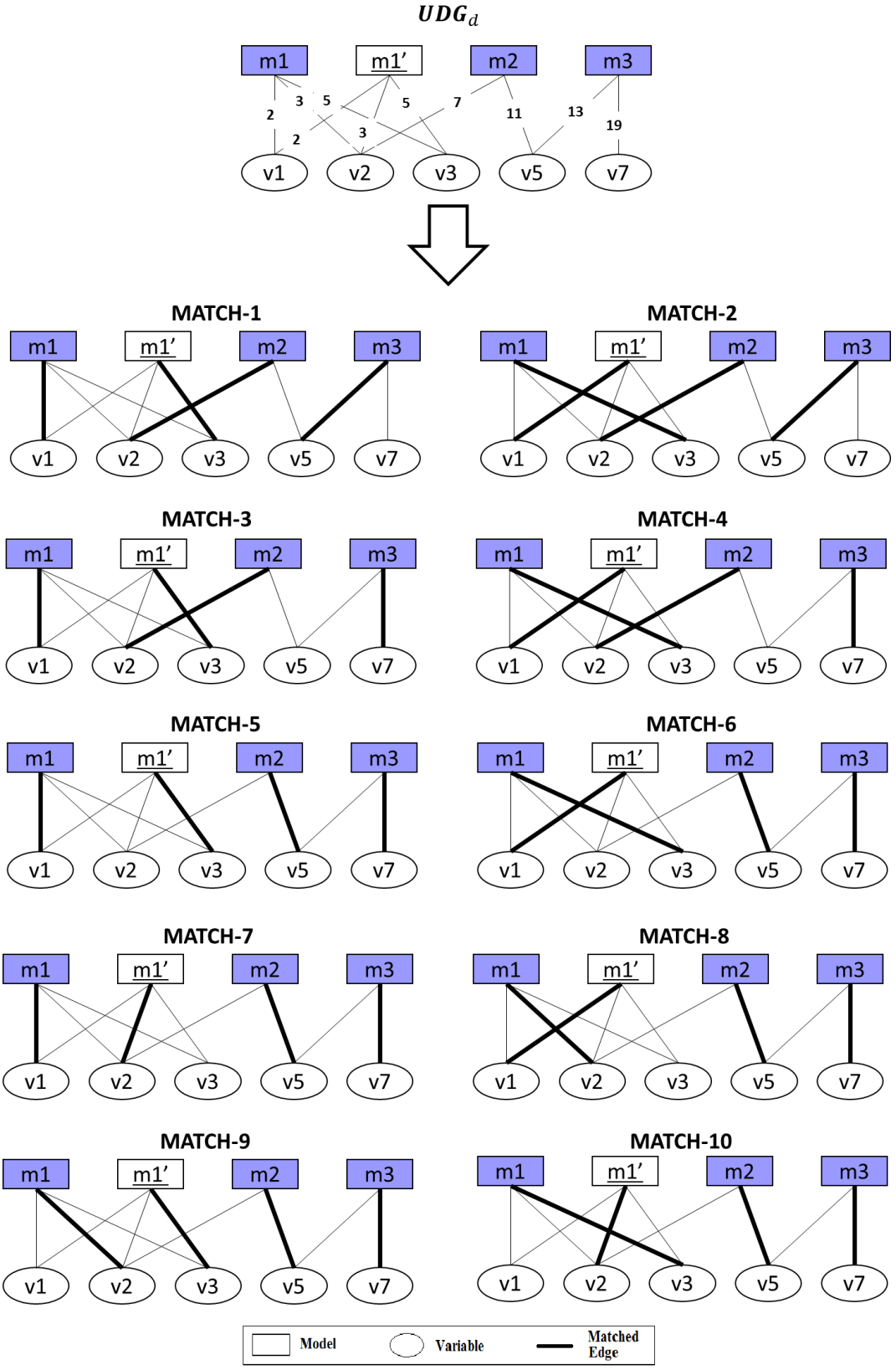


Figure 5.28: *Case-2: Enumeration of all possible matchings.*

variables in the corresponding workflow. For this, the reversal weights assigned to edges of the involved models as shown in Figure 5.23 are used. For the above filtered list of matchings, the sum of the reversal variables is calculated as shown in Table 5.11. It can be seen that matchings MATCH-1, 3, 5 and 9, have the same number of reversed variables i.e. 3. MATCH-7 has 4 reversed variables. Hence, any of the matchings i.e. MATCH-1, MATCH-3, MATCH-5, and MATCH-9 can be selected to create the workflow.

TABLE 5.10: Case-2: Matchings and their corresponding product of prime numbers.

Models / Matchings	m1	m1'	m2	m3	Product	Guess
MATCH-1	V1	V3	V2	V5	910	v7
	2	5	7	13		
MATCH-2	V3	V1	V2	V5	910	v7
	5	2	7	13		
MATCH-3	V1	V3	V2	V7	1330	v5
	2	5	7	19		
MATCH-4	V3	V1	V2	V7	1330	v5
	5	2	7	19		
MATCH-5	V1	V3	V5	V7	2470	v2
	2	5	13	19		
MATCH-6	V3	V1	V5	V7	2470	v2
	5	2	13	19		
MATCH-7	V1	V2	V5	V7	1482	v3
	2	3	13	19		
MATCH-8	V2	V1	V5	V7	1482	v3
	2	5	7	13		
MATCH-9	V2	V3	V5	V7	3705	v1
	3	5	13	19		
MATCH-10	V3	V2	V5	V7	3705	v1
	5	3	13	19		

5.4.3.3 Case-3: Over-determined System of Models

Let's assume that known parameters are v2, v4, v6, v7 and v8. The known and unknown variable sets are:

$$\mathbb{K} = \{v2, v4, v6, v7, v8\} \tag{5.37}$$

$$\mathbb{U} = \{v1, v3, v5\} \tag{5.38}$$

TABLE 5.11: Case-2: Matchings and their associated sum of reversal weights.

Models / Matchings	**m1**	**m1'**	**m2**	**m3**	**Sum**	**Guess**
MATCH-1	V1	V3	V2	V5	3	v7
	1	0	1	1		
MATCH-3	V1	V3	V2	V7	3	v5
	1	0	0	1		
MATCH-5	V1	V3	V5	V7	3	v2
	1	0	1	1		
MATCH-7	V1	V2	V5	V7	4	v3
	1	1	1	1		
MATCH-9	V2	V3	V5	V7	3	v1
	1	0	1	1		

Variable set V is the union of above two sets, that is,

$$V = \mathbb{U} \cup \mathbb{K} \tag{5.39}$$

Here,

$$TNvar - NIvar - Noutmode \implies 8 - 5 - 4 = -1$$

$$\therefore TNvar - NIvar - Noutmode < 0$$

Hence, the system of models for the given known parameters is over-determined.

Figure 5.29 shows a bipartite graph with known and unknown variables. At the bottom of the figure, a graph marked by red-dotted rectangle is the UDG. It shows assignment of PNs to its edges. On the left of Figure 5.30 is a graph, UDG_d, obtained adding necessary duplicate nodes (models), while on the right is a matching obtained applying the maximum matching enumeration algorithm on the UDG_d. Here, it can be seen that, there is a model node, m3 (shown by the red-dotted ellipse), which is not incident on any of the bold edges (matching). This is because of unavailability of a parameter to be matched with it. That is, all the parameters associated to this model are designated as known. To resolve this issue of the over-determinacy, the user is advised to remove some of the above-associated parameters from the list of known parameters, v7 and v8, to obtain a determined system of models. Then, the process of constructing the workflow is repeated.

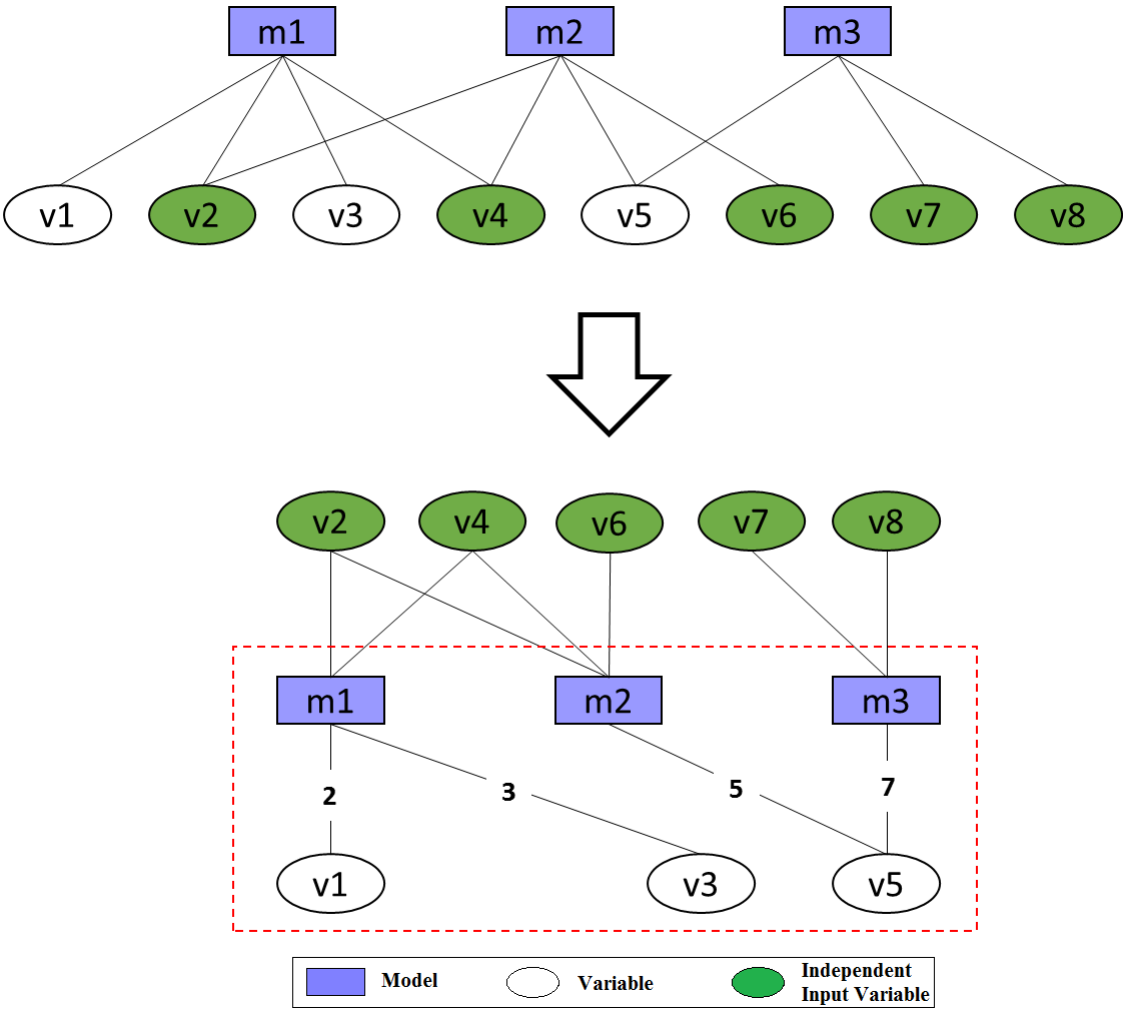


Figure 5.29: *Case-3: Known and unknown variables in a graph.*

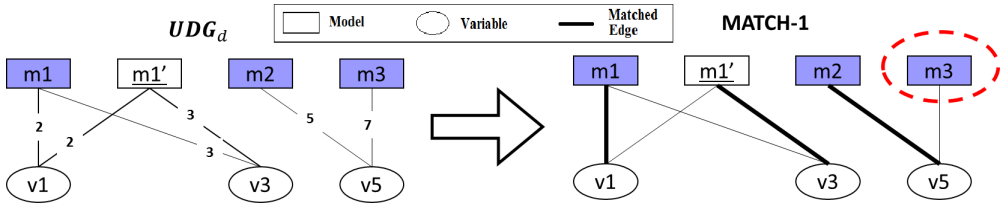


Figure 5.30: *Case-3: Enumeration of all possible matchings.*

5.5 Step-3: Complete Workflow Generation

In the previous steps, the sub-systems sequence (output from step-1), and the workflows (of individual sub-systems and system) (output from step-2) are obtained. In this step, a complete workflow, combining the sequence and workflows, is generated. The flow chart which explains the activities involved in this step is shown in Figure 5.31.

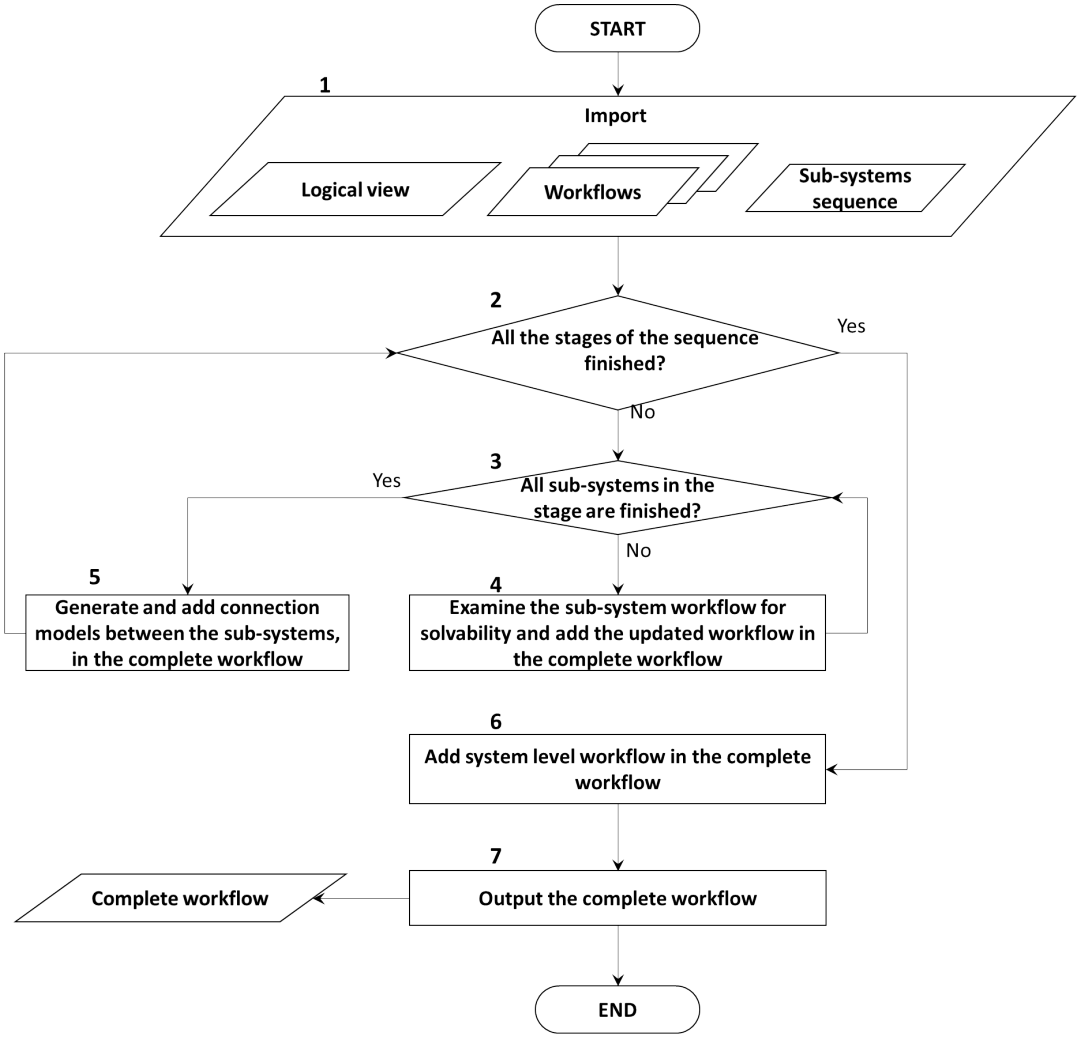


Figure 5.31: *Flow chart for step-3: complete workflow generation.*

The flow chart is briefly explained as follows:

START

1. **Logical view, sub-system sequence and workflows:** Initially, the system logical view, and the outputs, workflows and sub-system sequence, from the previous steps of the assessment process are taken as input.

2. **Decision - all the stages of the sequence finished?:** It is identified if all the stages in the sequence are finished. If the algorithm identifies the stages are finished then the method proceeds to step-6. Else, (if stages are not finished) then it proceeds to the next step.

3. **Decision - all the sub-systems in the stage finished?:** In this step, it is investigated if the workflows for all sub-systems in the current stage are already added in the complete workflow. If the answer is 'yes' then the control goes to step-5. Otherwise, the workflow for the next sub-system in the stage is examined in the next step.

4. **Examine the workflow for solvability:** Here, each of the sub-systems workflows are examined for requirement of an optimisation study for sizing the corresponding sub-system. Different sizing situations are possible depending on the types of a workflow (i.e. determined, under-determined and over-determined) and categories (based on whether the sub-system target variables are input or output in the workflow). Some require an optimisation study. More details about this step are given in Section 5.5.1. Wherever necessary, inputs from the designer are taken to set-up the optimisation study on top of the workflow to size the sub-system and added to the complete workflow. If the study is not needed then the workflow is added as it is in the complete workflow.

5. **Generate and add connection models between the sub-systems:**In this step, connection models between the sub-systems are generated and added in the complete workflow. For this, once the sub-systems in a stage are finished, the connection models between these sub-systems and their corresponding source subsystems are generated and added in the complete workflow, after the stage subsystems workflows.

6. **Add the system level workflow in the complete workflow:** In this step, a system level workflow is added to the complete workflow along with the aggregation

models that are necessary at the system level, and thus the final complete workflow is obtained.

7. **Output the Complete Workflow:** In this step, the complete workflow, which contains the workflows for sizing individual sub-systems along with optimisation studies if necessary and the system level workflow, is given as output.

END

The following sub-section explains in detail the significant task (of the flowchart) mentioned above i.e. examine the workflow.

5.5.1 Examine Workflow for Solvability

In this task, individual workflows of the sub-systems are examined in order to determine if an optimisation study (for more details see Chapter 2) is required to solve the sub-system sizing problem under consideration. A schematic of a part of a sub-system level logical view is shown in Figure 5.32. Here, the i^{th} sub-system has one input and one output port. The sink sub-system of the sub-system is shown on its right with a dotted lined rectangle. In the sub-system sequence, the sink sub-system is sized first, and therefore, the values of y_1^i and y_2^i are calculated at the sink sub-system and thus, their values are available. During step-2 of the architecture assessment process, workflows for all the sub-systems are produced. Depending on the determinacy (i.e. under-determined or determined) of the workflow and whether output port parameters are input or output in the workflow, there are four combinations possible for a sub-system workflow as shown in Figure 5.32. If the output port parameter is input to the workflow, the workflow is categorised as 'backward', otherwise, 'forward'.

The combinations contained in a workflow are described below.

- **Backward and determined:** In this combination, an optimisation study is not required to size (i.e. determine values of unknown variables including, also the parameters of sub-system components and/or input port(s)) the sub-system.

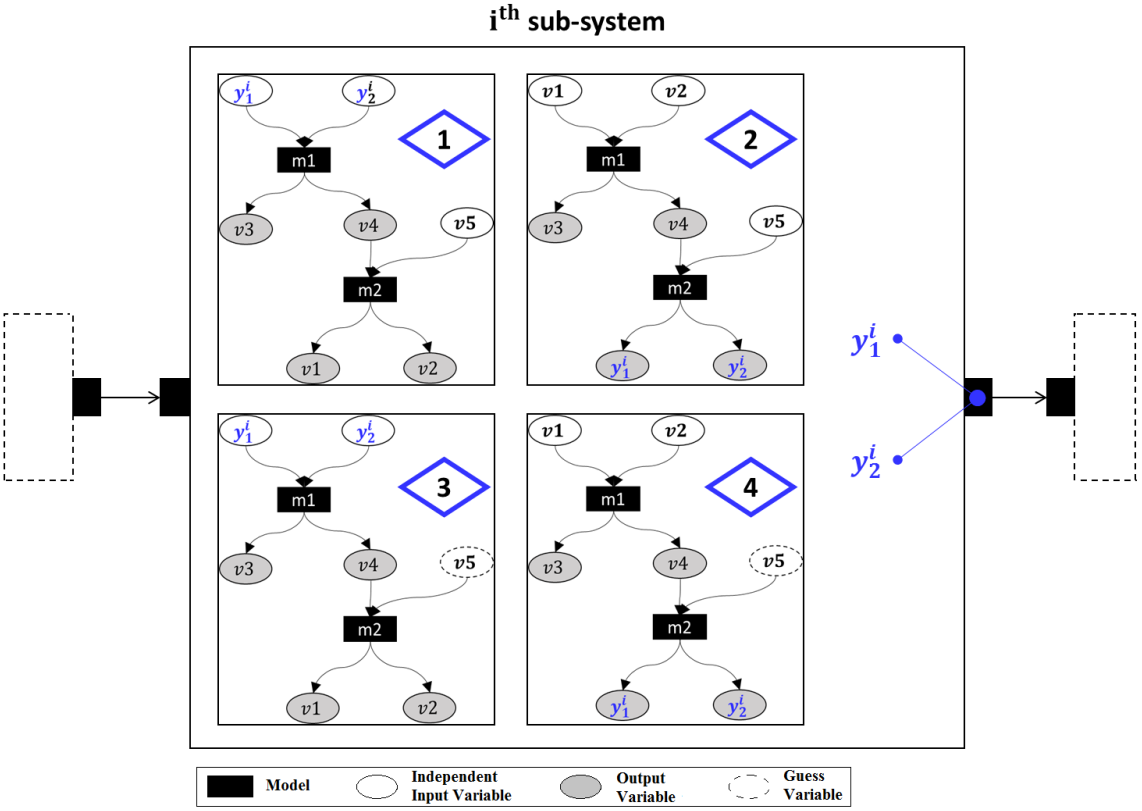


Figure 5.32: *Workflow types of a sub-system.*

- **Forward and determined:** If the workflow is classified as forward (meaning, output port variables are calculated in the workflow) and determined (meaning, all required (values of) inputs are supplied by the user), it is possible that calculated values of output port parameters may not be the same as that of the required values of these parameters (calculated at the sink sub-system). One way to meet the above requirement is to change workflow inputs supplied by the user within the respective user specified ranges. Here, this constitutes an optimisation study. If this combination is encountered, the user is asked to specify the inputs, their values (range) and (optimisation) objective to set-up the study. Given the above information by the user, the optimisation study is automatically set up. Here, in addition to the objective function provided by the user, additional function which minimises the absolute difference between the values of output port parameters and their corresponding values required at the sink is automatically added in the objective of the study.

- **Backward and under-determined:** In case of backward workflow, output-port

parameters (target parameters) are input to the workflow. Furthermore, the workflow is under-determined i.e. fewer parameter values than required were supplied by the user. As the target parameter values are input into the workflow, the requirement on them would be met. However, the user will need to specify ranges of 'guessed' values for a number of extra parameters, needed to make the workflow determined. To choose the optimal guess variable values, an optimisation study is required and is set up as described in the previous combination.

- **Forward and under-determined:** Similar to the above two combinations, this also needs a set-up of an optimisation study. The study tries to meet the requirement on the target variables changing the guess variables within their ranges specified by the user.

5.6 Summary

Presented in this chapter is a novel method for the assessment of system architectures, ultimately aiming to increase the efficiency and enable interactivity during architectural design space exploration. The assessment process contains three main steps: sequencing the sub-systems, constructing workflows (for sub-systems and system), and generating a complete workflow. It takes as input functional and logical views of the architecture and the steady-state computational models associated with each of the logical components. Several techniques have been proposed to support the assessment steps.

The technique developed to sequence the sub-systems is based on the source-sink relations between the sub-systems. To this purpose, a source-sink representation in the form of a DSM is produced using the description of the logical view. Then, the DSM sequencing algorithm is applied to obtain the sub-systems sequence. The technique is intended to produce the sub-systems sequence rapidly and automatically. This allows the user to dynamically and interactively modify the architecture and get the updated sequence.

In the workflow construction procedure, computational models for the logical connections and aggregated variables are dynamically produced. These models are then combined with the component models to form the computational sizing workflows of the sub-systems. Here, a workflow is modelled as a bipartite graph and a maximum matching enumeration algorithm is employed to produce all possible workflows. Duplicate model nodes are added to account for the models with more than one output. As a result, some of the matchings produced by the algorithm give identical workflows. A technique using prime numbers is proposed to filter out the matching producing identical workflows. The matching with the smallest number of reversed variables is selected from the remaining ones. This allows the user to select and construct the least computationally expensive workflow.

Finally, the enabler supporting generation of the complete workflow combines the outputs from the first two steps. Here, individual sub-system workflows are checked for the requirement of an optimisation study for the corresponding sub-system sizing. Furthermore, if there are coupled sub-systems in the sequence, they are sized simultaneously. The automated inspection of the individual workflows (to determine whether an optimisation study is required) and setup and execution of coupled sub-systems relieves the user from manually performing these tasks.

CHAPTER 6

Evaluation

Evaluation is an essential part of any research. The method developed (in the presented work) is a creation which involves various assumptions during its advancement. Considering the limited time, the research evaluation is confined to the initial stages of the Descriptive Study II (as described in Chapter 1). The study was conducted keeping in mind four objectives as follows: 1) to identify if the developed method can be employed to improve architectural design space exploration during conceptual design and has the expected effect i.e. reduction in manual activities, 2) to determine if the method indeed contributes to the improvement of the exploration process, 3) to identify any necessary improvement, and finally, 4) to evaluate the assumptions, based on which the method is developed.

6.1 Overview

Three types of evaluations are performed for the presented research. The scope of each of the evaluations within the 'impact model' is shown in Figure 6.1. These evaluations include 'Support', 'Application' and 'Success' evaluations. The 'Support' evaluation is conducted to check the consistency and inbuilt functionality of the developed methods. This evaluation is also referred to as 'verification' of the methods. The second type, 'Application' evaluation, was conducted to investigate if the method has the expected effect on the targeted 'Key factors'. In this research, the method is expected to reduce the number of manual activities (or increase automation) during the architecture definition and assessment processes. The final evaluation, 'Success' evaluation, was conducted to determine whether the methods indeed contribute to achieving the aim of the research. As it was not possible to employ the method on live industrial projects and investigate the effect on the quality of the architectural design space exploration (ADSE), experts' opinion is sought, regarding this evaluation.

To this purpose, a prototype object-oriented architecting tool, implementing all the proposed methods, was developed. This tool was integrated with AirCADia Architect, an in-house architecting tool developed at Cranfield during the TOICA project* for creating functional and logical views of the architecture. AirCADia Architect was extended in this research to include new methods that support both architecture definition and assessment processes and was employed during the evaluations.

The rest of this chapter is organised as follows. The next section describes the developed prototype software tool, followed by a description of a test-case (in Section 6.3), on which the methods are demonstrated. Results of the evaluations are presented in Section 6.4, and finally, summary and conclusions of the chapter are given in Section 6.5.

*Thermal Overall Integrated conceptualisation of Aircraft (TOICA)[13]

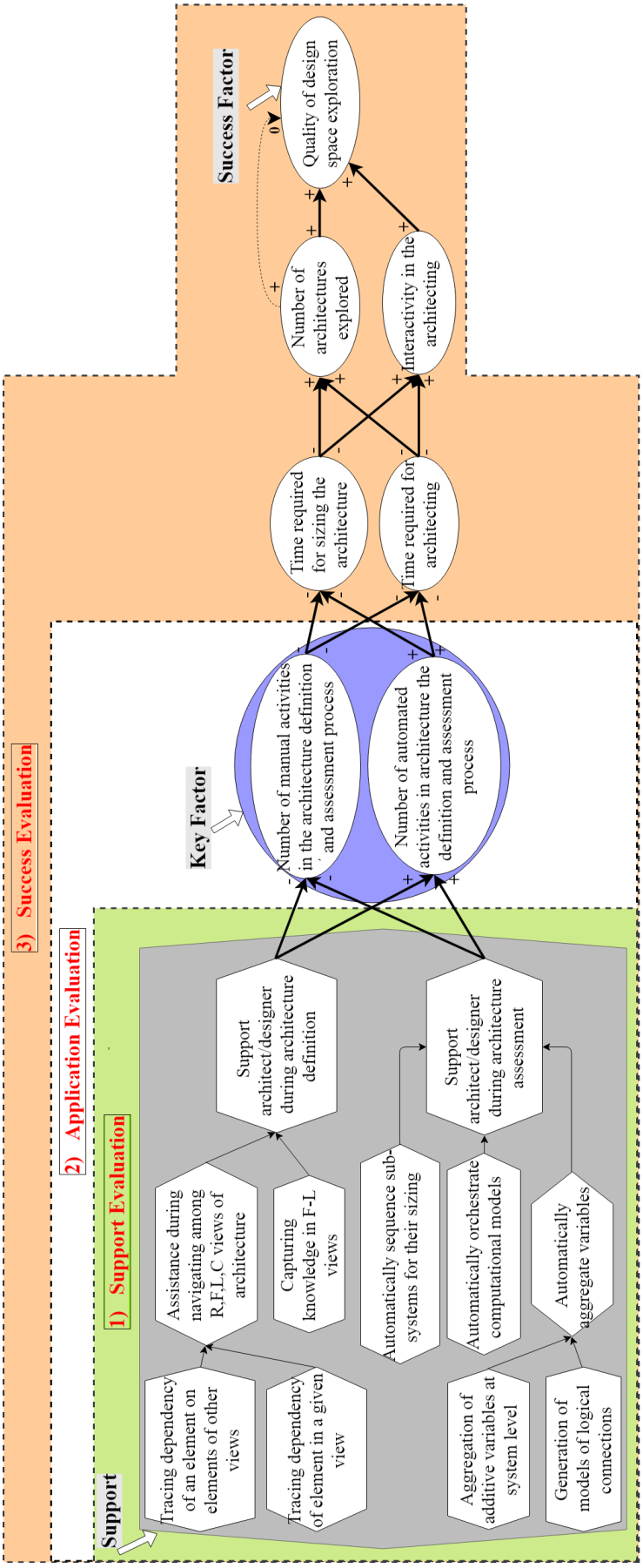


Figure 6.1: *Evaluation types and their scope.*

6.2 Computational Framework: a Software Prototype

As mentioned in Chapter 4, the work performed by Guenov et al.[44] is extended in this research, incorporating other views of the architecture. The focus of their work was on enabling the architect to interactively create functional and logical views of the architecture. The extension includes addition of new 'objects' and 'methods' to the existing architecting framework. The parts of the work which are not a direct contribution of the author are explicitly mentioned and presented here for completeness.

AirCADia Architect is implemented in an object-oriented programming language, C#. It integrates both definition and assessment of system architectures. The details of the structure of the tool and constituent modules are presented in Appendix C. Here, a diagramming library, 'GoDiagram' from Northwood Software[99] is employed to describe views of the system architecture.

6.2.1 Class Diagram

Here, a description is given of the objects that are required in the proposed architecting methodology. The proposed object model consists of several primary classes, including Project, Database, Architecture, Requirement, Function, Component, Port, Scenario, Parameter, Model, Workflow, Study, Treatment, and Sequence as described in Table 6.1. The high-level UML class diagram, describing the classes and their inter-relationships, is shown in Figure 6.2.

TABLE 6.1: Description of objects contained in the framework.

Class Name	**Description**
Architecture	Contains all the information that describes the architecture as a whole, including the list of all requirements, functions, component and their physical layout.
Complete workflow	Holds the complete information about the computational system sizing process.
Database	Stores the knowledge about all the architectures, categorised into requirements, functions, solutions, functional reasoning and so forth.
Function	Represents a function and have attributes and methods that enable decomposition of the function.
Model	Represents the mathematical equations which simulate the (physical) component's intended behaviour, function. It has (parameter) inputs and (parameter) outputs. The model object takes these input (parameter) objects and performs some operation on and produces out output (parameter) objects.
Parameter	Represents a physical quantity which describes the characteristics of a component or physical component. It holds numerical values of the quantities. Also, it is associated with models; in that, it is an input or output of a model.
Project	A container for both the database object and the list of architectures.
Requirement	Represents the stakeholder's needs which the system to be designed would fulfil. These could be of functional or performance type. It has attributes and methods enabling the decomposition of requirements, and their mapping to functions.
Scenario	Describes various scenarios of the system, listing active components of the system during the scenarios.
Sequence	Stores the sequence of the sub-systems for their sizing.
Solution	Stands for physical solutions implementing the functions.
Study	Represents the assembly containing workflow and/or models, and numerical treatments to conduct a design study
Treatment	Contains information of a particular mathematical treatment which is applied to model/workflow objects. For instance, an optimisation treatment applied to a workflow object.
Workflow	Describes a network of models aiming to find out the unknown parameters.

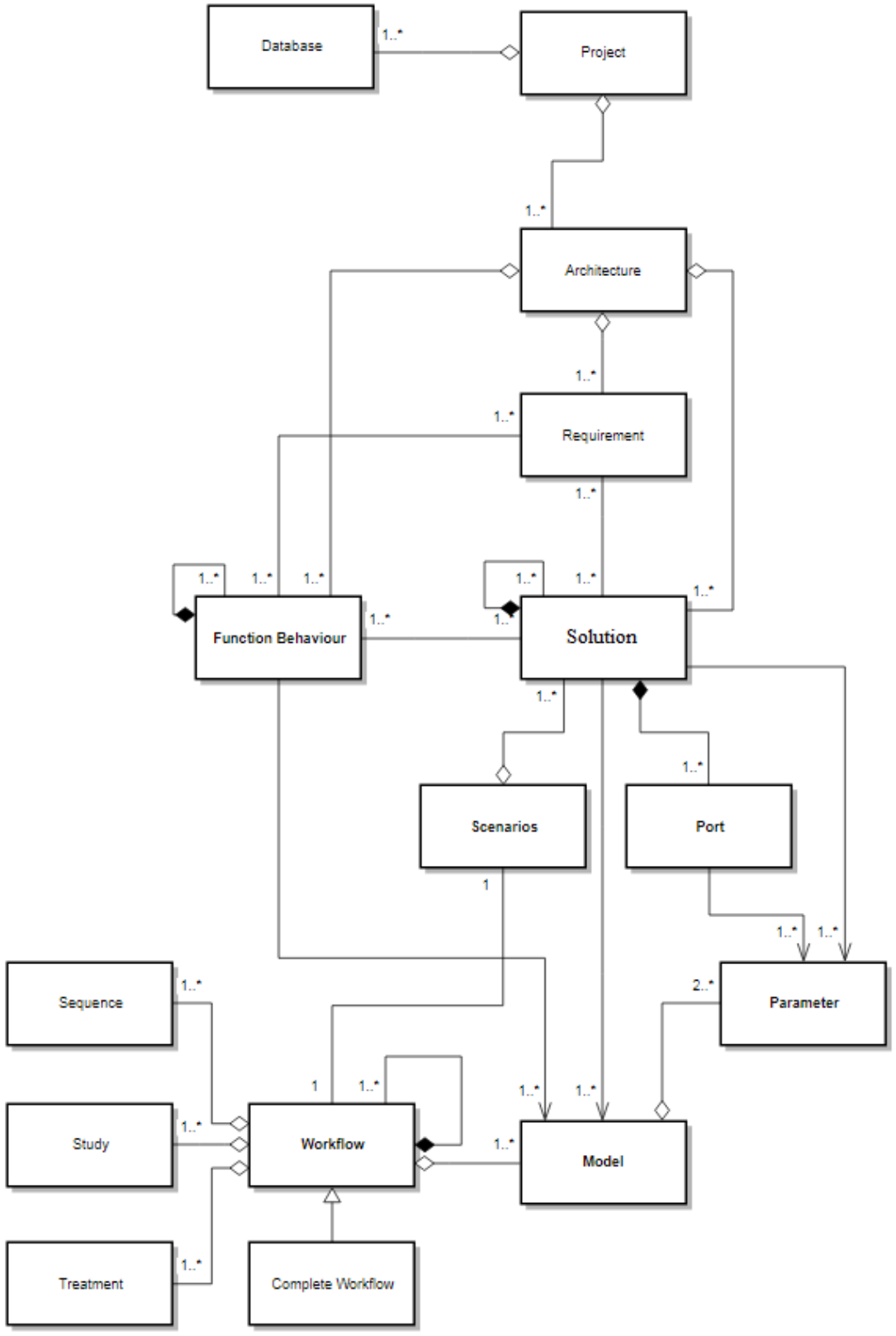


Figure 6.2: *UML class diagram of the proposed framework.*

6.3 Test-case

A test-case, based on transport aircraft design, was developed for application and success evaluation. It involves a hypothetical design scenario (see Figure 6.3), where, a conventional architecture is created, followed by its sizing, and system level performance evaluation. Then this architecture is modified to a more electrical one. The conventional aircraft architecture is converted to the more electrical one by modifying the ECS sub-system by employing electricity as a source of energy input to the ECS instead of bleed air. Finally, the new architecture is again sized and its performance is evaluated at the system level.

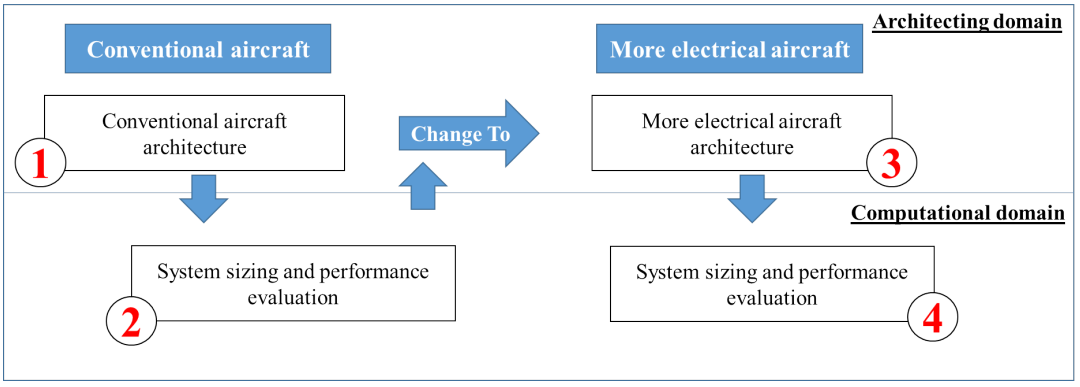


Figure 6.3: *Transport aircraft hypothetical design scenario.*

Only some of the sub-systems of the aircraft are taken into consideration in the test-case. The sub-systems considered are shown in Table 6.2. Here, only the ECS-components are modelled in detail. The steady state models considered for each of these components are shown in Appendix C.

TABLE 6.2: Aircraft sub-systems considered in the test-case.

Aircraft Sub-systems
Avionics (AVIO)
Cabin (CAB)
Electrical Power System (EPS)
Engine (ENG)
Environmental Control System (ECS)
Flight Control System (FCS)
Hydraulic Power System (HPS)
Ice-Protection System (IPS)
Pneumatic Power System (PPS)

6.4 Evaluation of the research

6.4.1 Support Evaluation

6.4.1.1 Architecture Definition

The enablers developed for supporting the architecture definition process are applied on a representative test-case, where the Environmental Control System (ECS) of the aircraft is synthesised, to verify the consistency and in-built functionality of the enablers.

The application of the prototype tool to the synthesis of an aircraft Environmental Control Systems (ECS) is presented. The discussion below covers how the ECS architectures are created, using both the notation used in Chapter 4, as well as relevant screen captures and descriptions of the software implementation.

A screenshot of the hierarchical view of the top-level requirements of the ECS, which contains both functional and performance requirements is shown in Figure 6.4. The two functional requirements which have associated explicit performance targets (constraints) are enclosed by dashed rounded-rectangles.

The top-level invariant function φ_1(‘provide comfortable environment to passengers’)

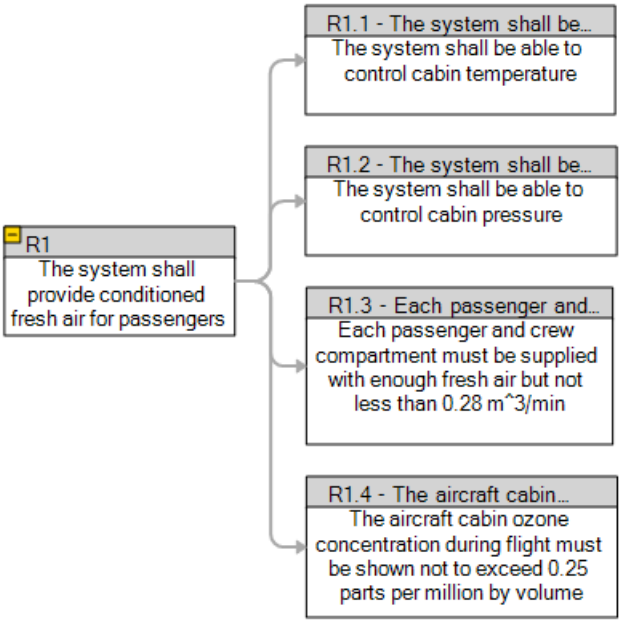


Figure 6.4: *Requirements (functional) hierarchical decomposition (prototype screenshot).*

was decomposed into four invariant sub-functions $\varphi_{1.1}$ ('source air'), $\varphi_{1.2}$ ('force airflow'), $\varphi_{1.3}$ (cool air') and $\varphi_{1.4}$ ('evacuate air'), as shown in Equation 6.1:

$$\varphi_1 \leftarrowtail \{\varphi_{1.1} \wedge \varphi_{1.2} \wedge \varphi_{1.3} \wedge \varphi_{1.4}\} \tag{6.1}$$

Next, solutions $\sigma_{1.i}$ were allocated to all four sub-functions $\varphi_{1.i}, \forall_{i=1,2,3,4}$. Two solutions were identified for the first function $\varphi_{1.1}$ (source air): $\sigma_{1.1.1}$ (pneumatic system), and $\sigma_{1.1.2}$ (ram air inlet), as stated in Equation 6.2. It is important to note that the symbol $\vee$ (logical OR) is used in Equation 6.2, because only a single solution (either $\sigma_{1.1.1}$ or $\sigma_{1.1.2}$) will be allocated to $\varphi_{1.1}$.

$$\varphi_1 \circ\!\!- \{\sigma_{1.1.1} \vee \sigma_{1.1.2}\} \tag{6.2}$$

The second function, $\varphi_{1.2}$ (force air), was mapped to $\sigma_{1.2}$ (electric fan), i.e. $\varphi_{1.2} \circ\!\!- \sigma_{1.2}$. Next, two solutions were identified for $\varphi_{1.3}$ (cool air): $\sigma_{1.3.1}$, (bootstrap air cycle machine) and $\sigma_{1.3.2}$ (vapour cycle machine), as stated in Equation 6.3.

$$\varphi_{1.3} \circ\!\!- \{\sigma_{1.3.1} \vee \sigma_{1.3.2}\} \tag{6.3}$$

Finally, the fourth function $\varphi_{1.4}$ (evacuate air) was mapped to $\sigma_{1.4}$ (outboard flow valve), i.e. $\varphi_{1.4} \circ\!\!- \sigma_{1.4}$. Note that the equations presented in this section are listed only for illustration of the architecture description using the notations described in Chapter 4. The actual architectural synthesis in the prototype tool involves only clicking, dragging, and dropping actions performed by the user.

After mapping solutions $\sigma_{1.i}$ to functions $\varphi_{1.i}$, the derived functions emerging from the chosen solutions, $\sigma_{1.i}$, can be identified and created. It is specified that solution $\sigma_{1.1.1}$ (pneumatic system) is utilized for $\varphi_{1.1}$ (source air), then two new derived functions, $\varphi_{1.5}$ (convert ozone) and $\varphi_{1.6}$ (demist air) are required. Alternatively, if solution $\sigma_{1.1.2}$ (ram air inlet) is utilized, then (in addition to $\varphi_{1.5}$ and $\varphi_{1.6}$) an extra function $\varphi_{1.7}$ (pressurize air) is required, since, unlike the air from the pneumatic system, which is already compressed, the ram air is not. The derived functions for the two solutions, $\sigma_{1.1.1}$ and $\sigma_{1.1.2}$, are expressed in Equation 6.4 and Equation 6.5, respectively.

$$\sigma_{1.1.1} \twoheadrightarrow \{\varphi_{1.5}, \varphi_{1.6}\} \tag{6.4}$$

$$\sigma_{1.1.2} \twoheadrightarrow \{\varphi_{1.5}, \varphi_{1.6}, \varphi_{1.7}\} \tag{6.5}$$

Next, solutions $\sigma_{1.5}$ (ozone converter), $\sigma_{1.6}$ (water separator) and $\sigma_{1.7}$ (compressor) were allocated to functions, $\varphi_{1.5}$, $\varphi_{1.6}$, and $\varphi_{1.7}$, respectively. Solution $\sigma_{1.5}$ (compressor) requires another derived function $\varphi_{1.8}$ (provide rotational energy), i.e. $\sigma_{1.7} \twoheadrightarrow \varphi_{1.8}$, which was mapped to solution $\sigma_{1.8}$(electric motor), i.e. $\varphi_{1.8} \circ\!\!- \sigma_{1.8}$. It should be noted, that technically the functional decomposition related to the handling of air ends with the identification of the compressor as a "leaf" node. Since functions $\varphi_{1.7}$ (pressurize air) and $\varphi_{1.8}$ (provide rotational energy) were derived from solution $\sigma_{1.1.2}$ (ram air inlet), solutions $\sigma_{1.7}$ (compressor) and $\sigma_{1.8}$ (motor) are relevant, as long as solution $\sigma_{1.1.2}$ is present. That is, if the architect decides to delete $\sigma_{1.1.2}$, the derived functions $\varphi_{1.7}$ and $\varphi_{1.8}$ (and their mapped solutions, $\sigma_{1.7}$ and $\sigma_{1.8}$) will automatically be deleted with the help of the elementary relations and data structures maintained in the object model.

The completed functional-hierarchical decomposition of the ECS is shown in Figure

6.5(a). The functional-logical zig-zagging is shown in Figure 6.5(b), where $\sigma_{1.1.1}$ (pneumatic system) and $\sigma_{1.3.1}$ (bootstrap air cycle machine) are used to fulfil $\varphi_{1.1}$ (source air) and $\varphi_{1.3}$ (cool air), respectively.

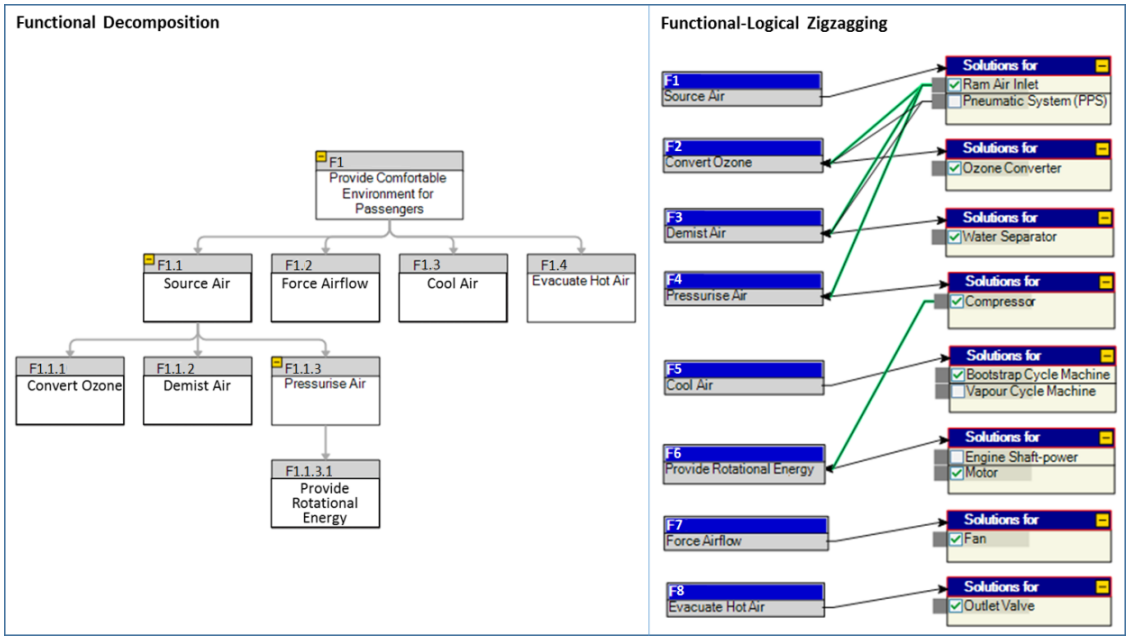


Figure 6.5: *Prototype screenshots of (a) functional-hierarchical decomposition view; (b) functional-logical zigzagging view.*

The green coloured lines in Figure 6.5 identify the (derived) functions which have been mapped to solutions in the logical domain.

Following the functional-hierarchical decomposition and the function-means zig-zagging, the next phase is to construct the functional and logical flow views. As shown in Figure 6.5(b), the functional reasoning model has two functions, i.e. $\varphi_{1.1}$ (source air) and $\varphi_{1.3}$ (cool air), which have more than one options to fulfil. In this case, $\sigma_{1.1.1}$ (pneumatic system) is selected to fulfil $\varphi_{1.1}$ (source air), and $\sigma_{1.3.1}$ (bootstrap air cycle machine) is selected for $\varphi_{1.3}$ (cool air). Once all the relevant functions and solutions are identified, the second step is to create the connections by linking functions and solutions through their ports. AirCADia Architect enables the designers to work either in the 'functional flow view' or in the 'logical flow view' and, while working in one of these, the other is updated accordingly to maintain consistency of information across both the functional and logical domains. The resulting functional and logical flow views of the conventional (bleed air) ECS architecture are shown in Figure 6.6.

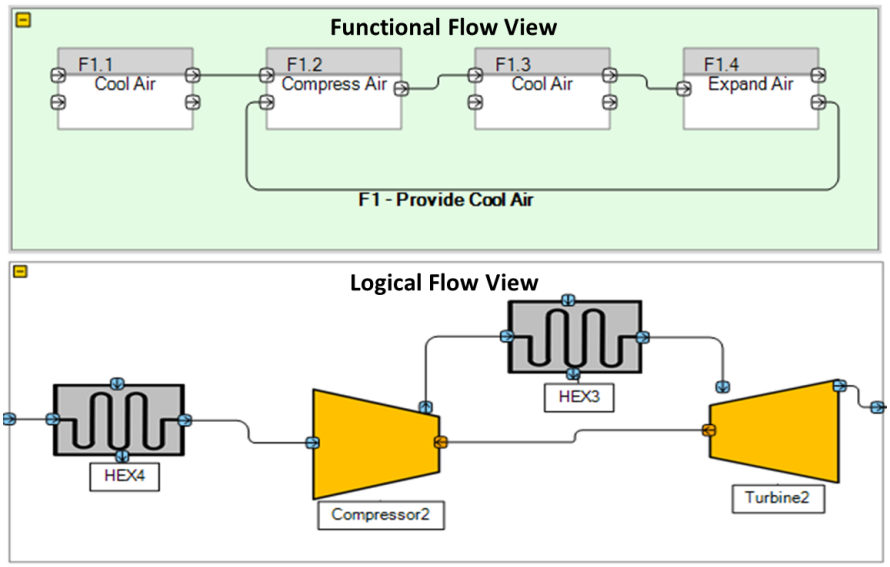


Figure 6.6: *Screenshots of the functional and logical views of the baseline ECS architecture.*

In order to increase the reliability of the ECS, two redundant solutions, $\sigma_{1.3.1}$ (i.e. two air cycle machine packs), could be employed to fulfil function $\varphi_{1.3}$ (cool air). The mapping of function $\varphi_{1.3}$ with two redundant solutions ($\{\sigma_{1.3.1} \wedge \sigma_{1.3.1}\}$) is represented by Equation 6.6.

$$\varphi_{1.3} \multimap \{\sigma_{1.3.1} \wedge \sigma_{1.3.1}\} \tag{6.6}$$

Suppose the designer wished to reduce the contribution of the ECS to fuel consumption (block fuel). The first task would be to identify which components or technologies of the existing baseline could be modified or replaced. The computational domain is used for this purpose. Here, Algorithm 3 is employed to identify the parameters (e.g. specific fuel consumption, wing's lift-to-drag ratio, (bleed) power off-take etc.) which affects the fuel consumption.

The next step is to identify the affected functions and solutions. Suppose the designer decided to replace solution $\sigma_{1.1.1}$ (pneumatic system) with $\sigma_{1.1.2}$ (ram air) for function $\varphi_{1.1}$ (source air)(here, the rationale is that the pneumatic power is inefficient, since a heat exchanger (pre-cooler) is used to cool the over-compressed and overheated bleed air, discharging excess energy back into the atmosphere as waste heat. The amount of wasted energy can reach up to 30% depending on the operating flight conditions[100]).

Algorithm 1 enables the architect to identify the affected functions and solutions by exploring the functional-logical (zigzagging) structure. In this case, functions $\varphi_{1.7}$ (pressurise air) and $\varphi_{1.8}$ (provide rotational energy) are the derived functions of $\sigma_{1.1.2}$ (ram air). Therefore, these two functions and their allocated solutions, $\sigma_{1.7}$ (compressor) and $\sigma_{1.8}$ (motor) need to be added to the functional and logical flow views of the electrical ECS architecture as shown in Figure 6.7. Algorithm 2 traces dependencies among the selected solution (motor) and the functions as highlighted in red in the figure.

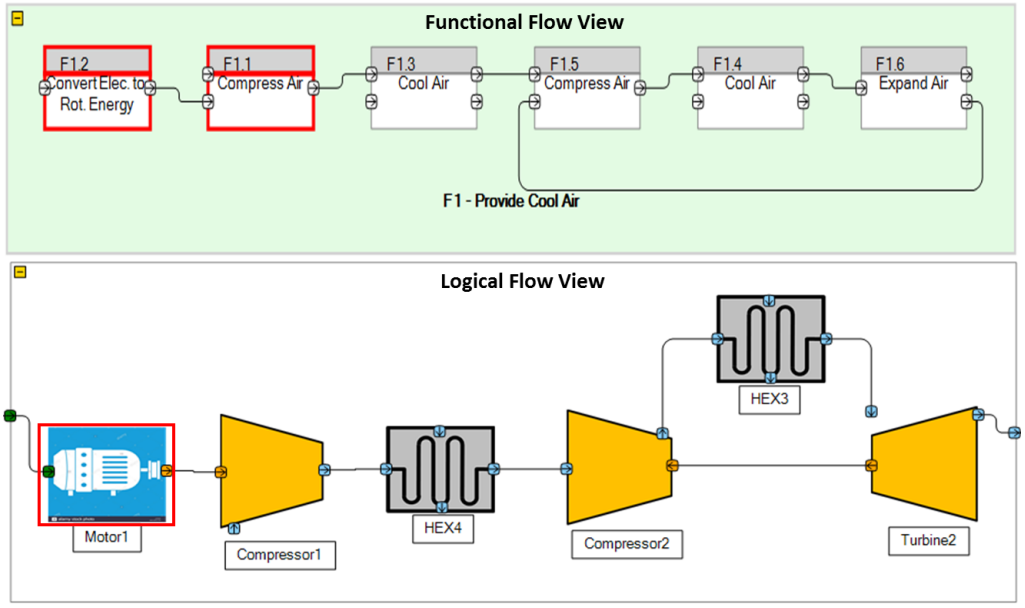


Figure 6.7: *Screenshots of the functional and logical views of the electrical ECS architecture.*

The enablers do support to capture and reuse the 'functional reasoning' knowledge during the architecting process and trace the dependency of architectural elements. Results from the application of the enablers on the test-case confirmed the correctness of the enablers as well as their inbuilt functionality.

6.4.1.2 Architecture Assessment

As the proposed (assessment) enablers overcome some of the limitations of the existing methods in the current research context, the comparison is performed for cases where both the existing and proposed methods are applicable. For the other conditions (where existing methods are not valid), the results from the proposed method are compared with

the results obtained by manually performing the relevant tasks on a simplified test-case. The latter is simplified because it is difficult for a human to perform the (sequencing) tasks and to check the correctness of the results of a complex case.

6.4.1.2.1 Sequencing of sub-systems

As mentioned earlier, the 'support' evaluation focuses on verifying the correctness of the developed enablers. Therefore, the enablers were investigated to confirm whether they provide the expected results for a given logical view. It should be emphasized at this point that, ideally, the proposed method should be compared with a sizing sequence produced by an equivalent (competitor) method or with a result from an ongoing or a completed industrial program. To the author's best knowledge, there is no existing analogue of the proposed method. Also, a design sequence from an industrial program is not available at present. For this reason, the sequencing result is compared with the sequence proposed by Liscouët-Hanke et al.[47].

Recall that a system's logical view is taken as input in the first step (see Chapter 5) of the sequencing process. To this purpose, a number of aircraft subsystems were considered in this test-case, in addition to those listed in Table 6.2, and are listed in Table 6.3. Here, more sub-systems are considered to demonstrate the sequencing enabler on a realistic test-case representing a design of a complex system containing a large number of sub-systems. The corresponding logical view created in the prototype software tool described in Section 6.2, is shown in Figure 6.8. This view is given as input to the enabler which sequences the sub-systems. For this purpose, a source-sink DSM (as shown in Figure 6.9) is extracted from the logical view and subsequently, sequenced. The sequence obtained is shown in Figure 6.10. The stages represent the sequential order (from left to right) in which the sub-systems should be sized.

From the sequence in Figure 6.10, it can be seen that the consumers (CAB, IPS, FPS etc.) are placed in the first stage, whereas sources (ENG, APU, EPS etc.) are assigned to the last stage. Transformers (HPS, PPS etc.) are placed in the stages in between the above two. As per Liscouët-Hanke et al.[47], the sub-systems of the power systems

group are sized in the following order: power consuming sub-systems, power distribution and transforming sub-systems, and finally power generation sub-systems. The proposed method reproduces the same sequence.

TABLE 6.3: Aircraft sub-systems considered in the test-case.

ATA Number	Sub-system name
ATA 21	Environmental Control System (ECS)
ATA 23	Communications
ATA 24	Electrical Power System (EPS)
ATA 24	Cabin (CAB)
ATA 27	Flight Control System (FCS)
ATA 28	Fuel System
ATA 29	Hydraulic Power System (HPS)
ATA 30	Ice Protection System (IPS)
ATA 32	Landing Gear (LG)
ATA 33	Lightings
ATA 34	Navigation
ATA 36	Pneumatic Power System (PPS)
ATA 38	Water and Waste
ATA 49	Auxiliary Power Unit (APU)
ATA 72	Engine (ENG)

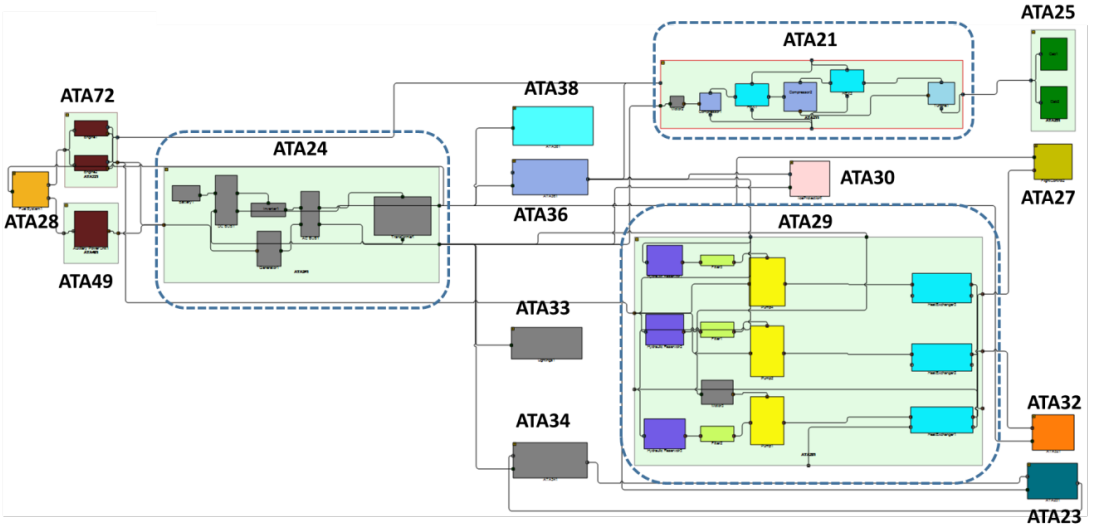


Figure 6.8: *Logical view of the aircraft architecture.*

		Sink →	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		Source ↑	21	23	24	25	27	28	29	30	32	33	34	36	38	49	72
1	ECS	ATA 21	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
2	Communication	ATA 23	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0
3	EPS	ATA 24	1	1	1	0	1	1	1	1	1	1	1	1	1	0	0
4	CAB	ATA 25	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
5	FCS	ATA 27	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
6	Fuel System	ATA 28	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1
7	HPS	ATA 29	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0
8	IPS	ATA 30	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
9	LG	ATA 32	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
10	Lightings	ATA 33	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
11	Navigation	ATA 34	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0
12	PPS	ATA 36	1	0	0	0	0	0	1	1	0	0	0	1	0	0	0
13	Water&Waste	ATA 38	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
14	APU	ATA 49	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0
15	ENG	ATA 72	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1

Figure 6.9: *Source-sink DSM of the logical view.*

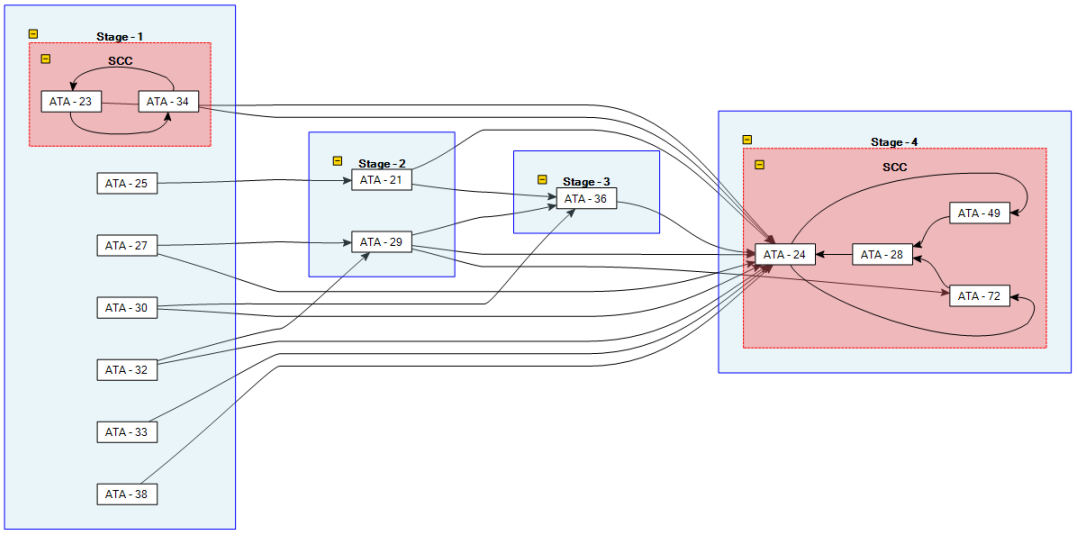


Figure 6.10: *Sub-system sizing sequence for the given logical view.*

6.4.1.2.2 Construction of the Workflow

To verify the correctness of the proposed method, it is compared with the existing methods such as the one devised recently by Balachandran[18]. It should be emphasized that Balachandran's algorithm requires a determined system of models to operate on. In the case of an under-determined system of models, a single computational workflow is created by randomly choosing the input variables from the set of (default workflow)

independent inputs. By contrast, the proposed method generates all possible workflows for the under-determined system of models.

Both methods are supplied with the same set of computational models, and the results are compared. There are some cases, where Balachandran's method[18] does not work because of its inability to deal with a system of models where variables can be outputs from more than one model. In those cases, the correctness of the workflows is manually checked.

The evaluation of this enabler is divided into two parts: 1) Comparison of the proposed method with Balachandran's method[18] in the occasions where both methods are valid, and 2) Manual verification of the proposed method for the circumstances other than considered in the first part.

Comparison with Balachandran's method[18]: The proposed method is compared with Balachandran's method[18] on the following features:

- Computational workflow structure/topology results, obtained for the inputs in the following three cases:
  - Determined (the number of independent variables is equal to the number of variables required for solving the models)
  - Under-determined (the number of independent variables is smaller than the number of variables required for solving the models)
  - Over-determined (the number of independent variables is greater than the number of variables required for solving the models)

For this part of the evaluation, a system of models is considered as shown in Figure 6.11. The models' default inputs and outputs are shown in Table 6.4.

As per Balachandran[18], the determinacy of a system of models is checked by the following conditions.

$$TNvar - NIvar - Noutmode = 0 \quad \text{Determined} \tag{6.7}$$

$$TNvar - NIvar - Noutmode > 0 \quad \text{Under-determined} \tag{6.8}$$

$$TNvar - NIvar - Noutmode < 0 \quad \text{Over-determined} \tag{6.9}$$

where, $TNvar$ - Total number of variables

$NIvar$ - Number of independent variables (i.e. known variables)

$Noutmod$ - Sum of the total number of outputs of each model in a system

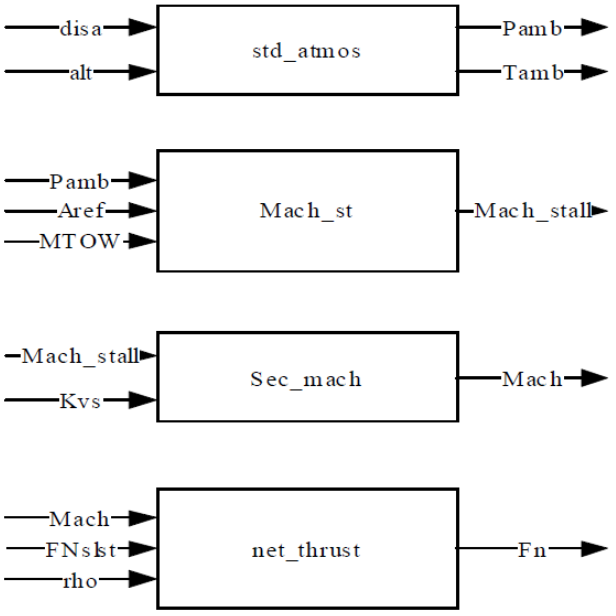


Figure 6.11: *A system of computational models*[18].

TABLE 6.4: Default workflow inputs and outputs.

Independent Inputs	Outputs
1. disa	1. Pamb
2. alt	2. Tamb
3. Aref	3. Mach
4. MTOW	4. Mach_stall
5. Kvs	5. Fn
6. FNslst	
7. rho	

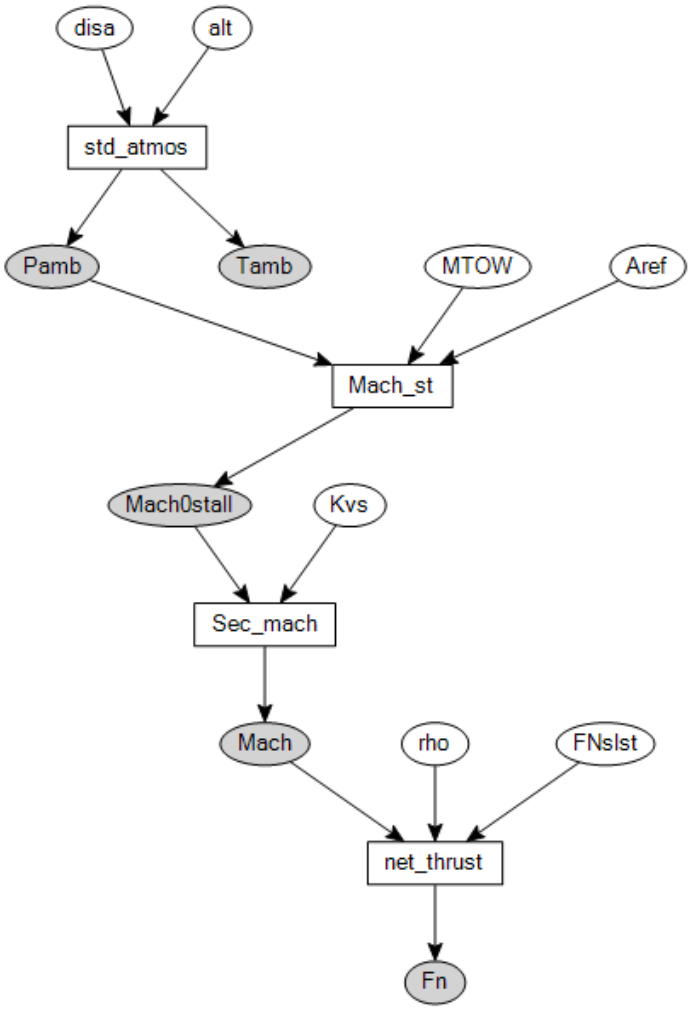


Figure 6.12: *Default workflow.*

Case-1: Determined system of models

The default workflow satisfies the condition in Equation 6.7.

Here, $TNvar = 12$, $NIvar = 7$ and $Noutmod = 5$, and as per the Equation 6.7,

$$12 - 7 - 5 = 0$$

Let's take a new set of independent inputs as shown in Table 6.5.

Again, the number of independent variables is 7. The condition in Equation 6.7 is satisfied; therefore, the system of models is determined. The same set of models and independent inputs are given to both, the proposed method and Balachandran's method[18] and the obtained workflows are compared. It is found that the workflows obtained by

TABLE 6.5: Determined case: a new set of independent inputs.

Independent Inputs
1. Tamb
2. alt
3. Aref
4. MTOW
5. Kvs
6. FNslst
7. rho

both the methods are same. The new workflow produced by both the methods is shown in Figure 6.13. It should be noted that, in this workflow, the model *std_atmos* has a different set of inputs and outputs than its default condition shown in Figure 6.12. That is, the model is a reversed. Such models require a numerical treatment to reverse them, as described in Chapter 2.

Discussion:

Balachandran's method[18] checks the determinacy of the system of models given a number of independent inputs and does not pay any attention to the independent variables themselves. That is, only the number of independent variables is not sufficient to check the determinacy of the system as there could be locally over-determined models for the given set of independent inputs. For instance, for the set containing 7 independent inputs rho, disa, alt, Aref, MTOW, Kvs, and Pamb, the condition in Equation 6.7 is still satisfied, but the model *std_atmos* is overdetermined. This model has two inputs and two outputs in the default condition, as shown in Figure 6.12. However, with the above set of independent inputs, for this model, the independent inputs are three, unlike in its default condition. Therefore, the expression in Equation 6.7 is a necessary, but not sufficient condition to check the determinacy of a system of models. The above issue is tackled by Datta[17] in his thesis; however, his method, an extension of Balachandran's method[18], also requires a determined system of models at the start.

In the presented method, the user is advised in the above occasions to relax some of the constraints on the locally over-determined models (for more details see Chapter 5). That is, the list of relevant (to the model) variables from the independent variables set is displayed, and the user is asked to remove the variables over-constraining the model

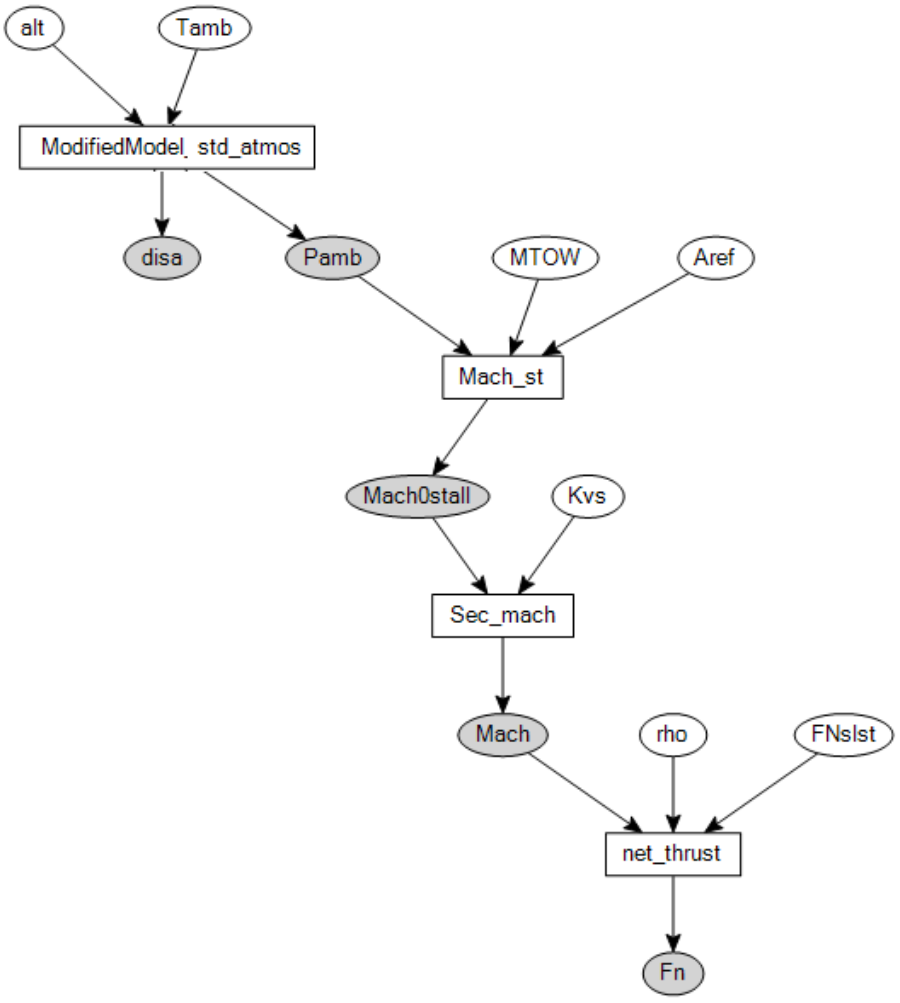


Figure 6.13: *Workflow with a new set of independent inputs.*

from the set of independent variables. This is discussed in more detail below, when comparing the method in case of an overdetermined system of models (Case-3).

Case-2: Under-determined system of models

The default workflow satisfies the expression in Equation 6.8. The independent inputs are as given in Table 14. Here, $TNvar = 12$, $NIvar = 6$ and $Noutmod = 5$, and as per the Equation 6.8,

$$12 - 6 - 5 = 1 > 0$$

For an under-determined system of models, Balachandran's method[18] produces a single workflow given a determined workflow of these models. However, the proposed method generates all the possible workflows as shown in Figure 6.14, Figure 6.15 and Figure 6.16. These workflows also contain the workflow produced by Balachandran's

method[18]. In some of them, there are model nodes (i.e. rectangles) with a text 'ModifiedModel'. These are reversed models. These models need a numerical treatment to solve them as their inputs and outputs are different from their default condition.

TABLE 6.6: Under-determined: a set of independent inputs.

Independent Inputs
1. disa
2. alt
3. Aref
4. MTOW
5. FNslst
6. rho

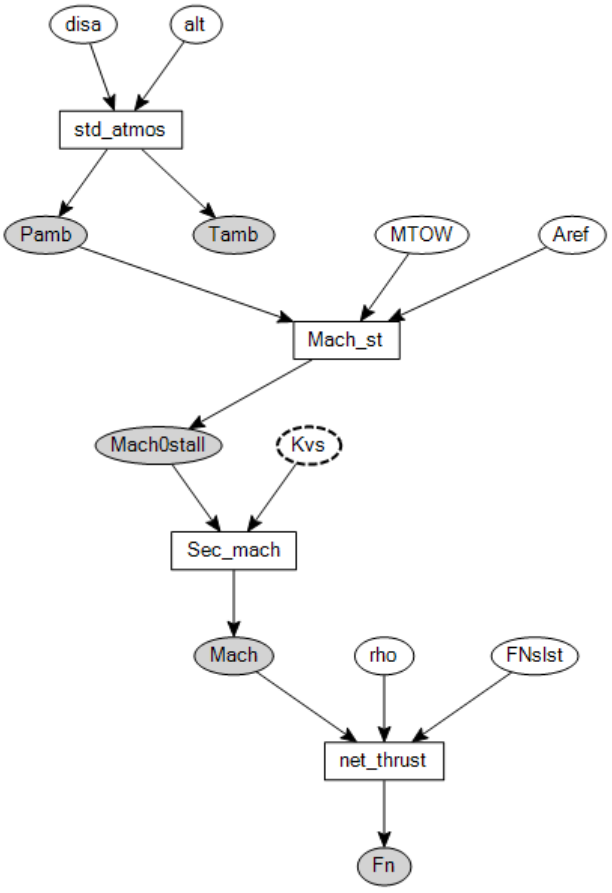


Figure 6.14: 1st *under-determined workflow.*

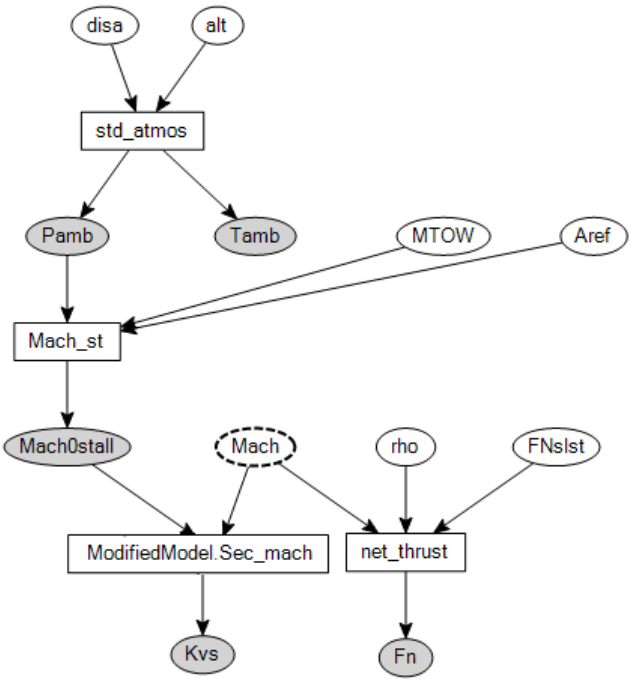


Figure 6.15: *2nd under-determined workflow.*

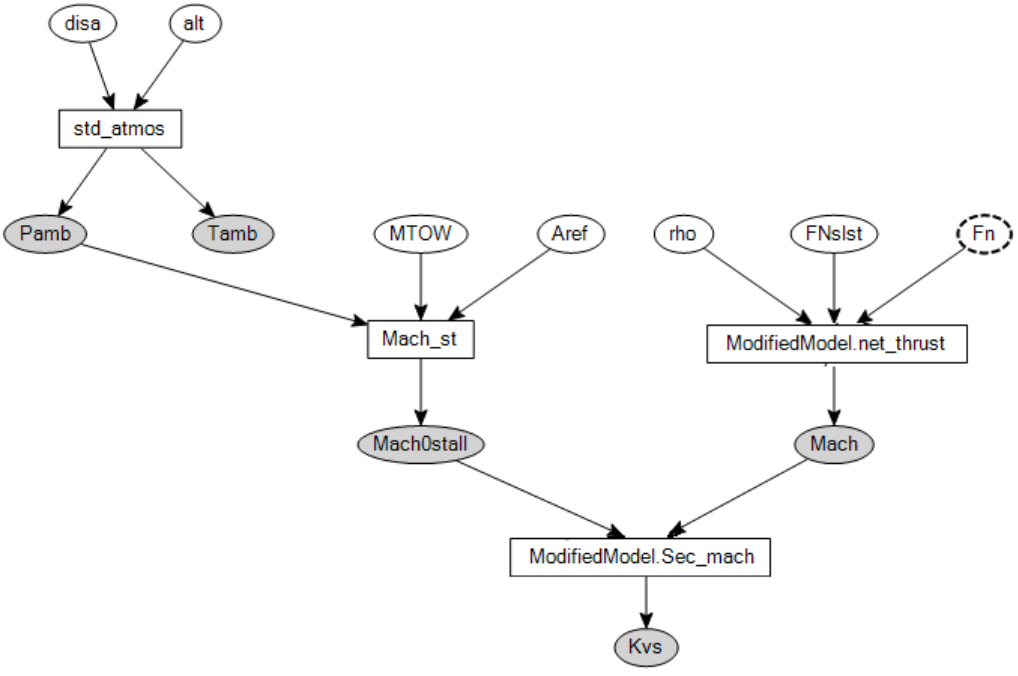


Figure 6.16: *3rd under-determined workflow.*

Discussion:

Although the proposed method produces all possible workflows, currently, there is no way to choose a preferred workflow out of the available workflows. Furthermore, the

way the method works, it produces duplicate workflows. There is a need to filter out the duplicate matchings and list only distinct workflows and sorts these with minimum reversed variables at the top. This would assist the user in choosing the preferred workflow. The enablers proposed for filtering out duplicate matchings, and selecting a preferred workflow have already been verified through similar illustrative examples in Chapter 5 and are not verified again to avoid repetition. Furthermore, this verification is not within the scope of comparison with Balachandran's method[18].

Case-3: Over-determined system of models

The default workflow satisfies the condition expressed in Equation 6.9. The independent inputs are as given in Table 6.7.

Here, $TNvar = 12$, $NIvar = 8$ and $Noutmod = 5$, and as per the Equation 6.9,

$$12 - 8 - 5 = -1 < 0$$

In the case of the over-determined system of models, Balachandran's method[18] does not produce any workflow. The proposed method does not produce any workflow either. However, the user is prompted with the list of over-determined models and asked to remove some of the independent inputs associated with the over-constrained models, so that these models become determined.

TABLE 6.7: Over-determined: a set of independent inputs.

Independent Inputs
1. disa
2. alt
3. Aref
4. MTOW
5. Kvs
6. FNslst
7. rho
8. Pamb

Discussion:

While dealing with a large number of models, it is difficult for a user to manually find the locally over-determined models of the system. In the proposed method, assistance

is provided by listing the over-determined models and their associated variables that would (potentially) need to be removed from the set of independent inputs in order to resolve the over-determined system.

Manual verification

This part of the evaluation is included because the method needs to be verified in the cases (a variable is an output from more than one model) where existing methods are not applicable or available. For this purpose, the author created computational workflows for a small set of computational models. The results obtained are then compared with the results obtained by the proposed computational method. The models considered are shown in Figure 6.17. Here, it can be seen that variable, 'd' is output from two different models, M1 and M2, whereas variable, e is output from models, M1 and M3.

Let us assume that variables 'a' and 'b' are known. The workflow obtained by the proposed method is shown in Figure 6.18. There is only one workflow possible for the given number of known variables. The author manually obtained the same workflow. Now consider the under-determined case by reducing the number of known inputs (less than two). In this case, there is only one known input (i.e., variable, g). The method produced six possible workflows for this input. The author could not derive all of these, but could verify the correctness of the (six) ones produced by the computer.

Discussion:

In the cases where the variables are outputs from more than one model, it is possible to find the number of independent inputs (i.e., known variables) by using Equation 6.7 . For instance, the total number of variables (TNvar) in the system in Figure 6.17 is eight and total number of outputs from the models is six. Putting these numbers in Equation 6.7 gives two independent inputs:

$$NIvar = TNvar - Noutmod = 8 - 6 = 2$$

Event for this relatively simple test case the computer outperformed the human by finding more workflows. As mentioned in the first part of the evaluation, ranking according

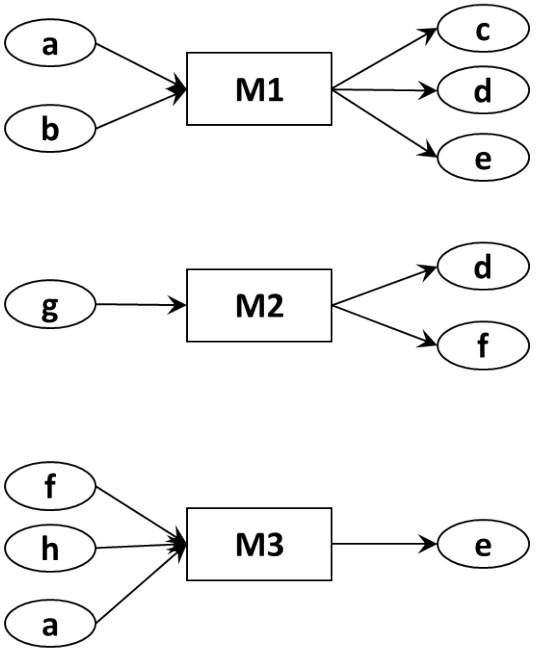


Figure 6.17: *Computational models with one variable output of two models.*

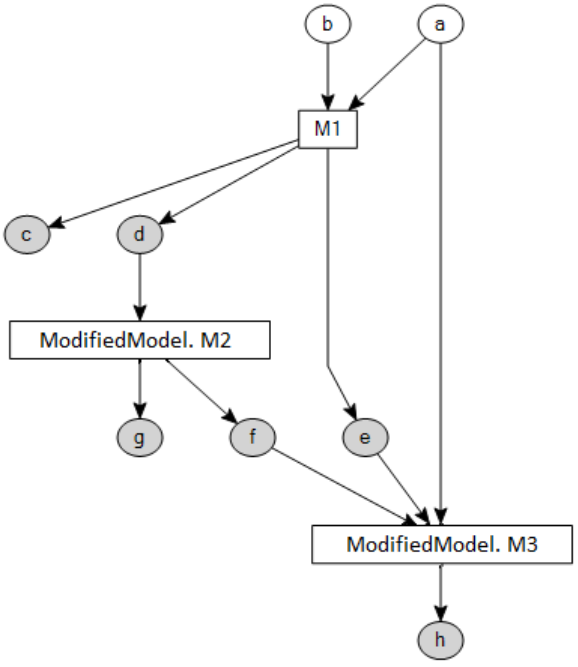


Figure 6.18: *Workflow obtained with 'a' and 'b' as known variables.*

to the number of reversed models will help the user to choose the workflow with minimum reversed models. This is important, as solving of reversed models needs a numerical treatment, which results in increased computational cost. The proposed enablers for filtering out duplicate matchings and for ordering these were verified on representative test-cases in the Chapter 5, and therefore, are not verified again on the above system of models.

6.4.1.2.3 Generation of logical connection models and aggregation of additive variables

The evaluation of the methods developed for generating connection models and aggregating additive variables are trivial and are not verified explicitly. The method for generating connection models is demonstrated through the logical view of a simplified Environmental Control System (ECS) architecture. The latter is created in AirCADia architect, as shown in Figure 6.19. Here, all the logical connection arrangements are 'Type-I' (one-to-one) category. For each connection, depending on a number of associated port-variables, there can be more than one computational model as shown in Table 6.8. In the table, the port-parameters, $\dot{m}, P, T$, and Pow are mass flow rate, pressure, temperature and rotational power, respectively.

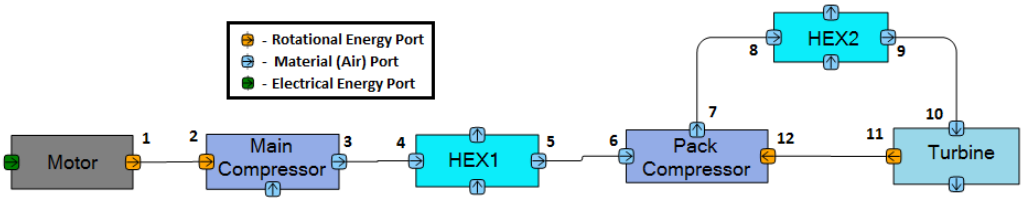


Figure 6.19: *ECS logical view.*

TABLE 6.8: Connection models.

Connections	Models
1-2	$Pow1 = Pow2$
3-4	$\dot{m}3 = \dot{m}4$ $P3 = P4$ $T3 = T4$
5-6	$\dot{m}5 = \dot{m}6$ $P5 = P6$ $T5 = T6$
7-8	$\dot{m}7 = \dot{m}8$ $P7 = P8$ $T7 = T8$
9-10	$\dot{m}9 = \dot{m}10$ $P9 = P10$ $T9 = T10$
11-12	$Pow11 = Pow12$

6.4.2 Application Evaluation

The purpose of this evaluation was to confirm that the proposed method reduces manual activities by increasing automation during both architecture definition and assessment processes.

For the evaluation purposes, the method is applied on the test-case described in Section 6.3. Ideally, the author should have re-created the test-case in the existing tools and compared the involved efforts in performing the hypothetical design scenario with the proposed approach. However, project resources and time were not available, and therefore an in-depth comparison falls out of the scope of this research.

Hence, the method is compared to selected relevant existing approaches. For example, on the architecture definition side, the application of the proposed methods is compared with architecting tool, based on the well-known architecting language, *SysML*, and on the assessment side, is compared with the *OpenModelica*. It should be emphasised that the proposed approach is compared with these approaches based on the functionality only with respect to the tasks in the test-case.

The results of the application of the method on the test-case are described in detail in Appendix C. The logical view of the conventional architecture created in the prototype

tool is shown in Figure 6.20. As described in the hypothetical design scenario of the test-case in Section 6.3, this logical view is modified to a more electrical architecture as shown in Figure 6.21.

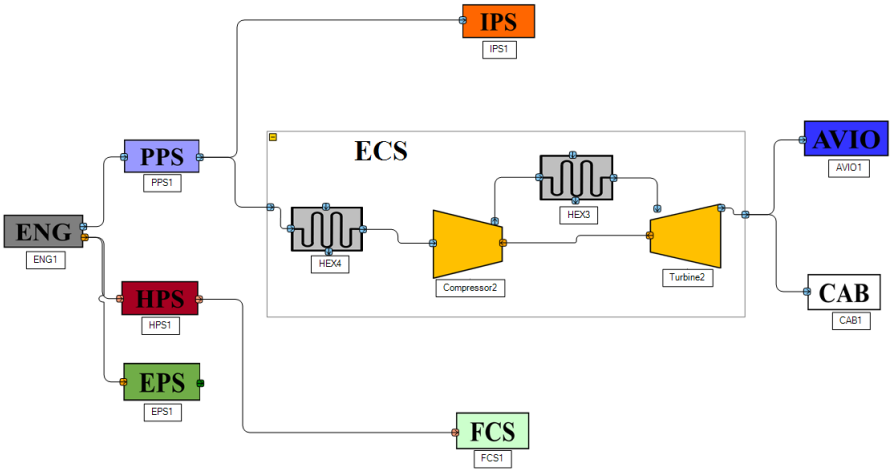


Figure 6.20: *Conventional aircraft architecture logical view.*

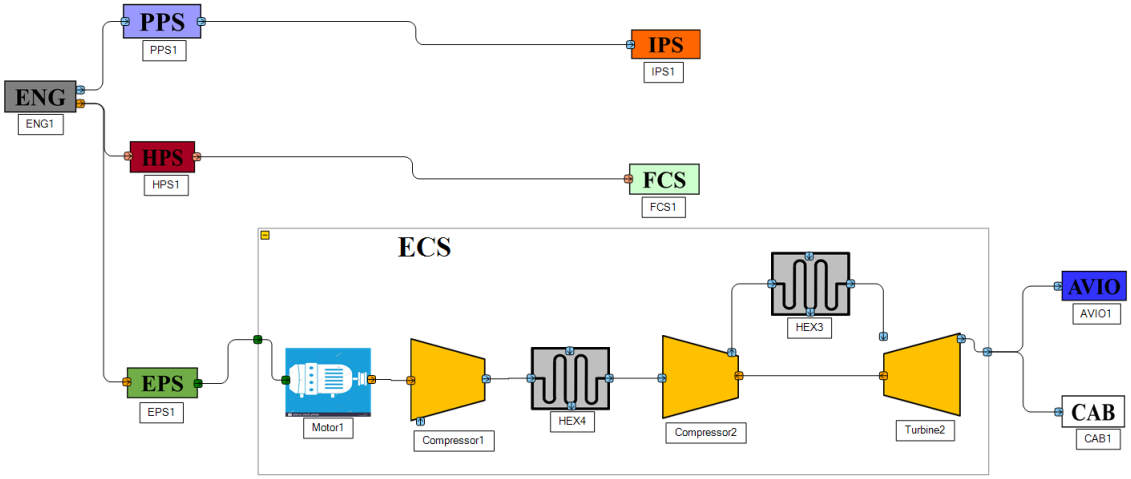


Figure 6.21: *More electrical aircraft architecture logical view.*

Table 17 summarises the comparison of the proposed method with SysML and OpenModelica with respect to the major involved tasks in the test-case. It can be seen that the proposed method indeed reduces the manual tasks involved in the architecture definition and assessment processes.

TABLE 6.9: Comparison of the proposed method with SysML and OpenModelica

✗ = manual, ✓ = automatic, NA = not applicable or does not support		
Architecture definition process		
Task	**SysML based tool**	**Proposed method**
Traceability between elements of different views	✗	✓
Traceability within functional reasoning	✗	✓
Traceability within computational view	NA	✓
Functional reasoning knowledge capture	NA	✓
Architecture assessment process		
Task	**OpenModelica**	**Proposed method**
Sequencing of sub-systems	✗	✓
Construction of workflows	✓	✓
Complete workflow construction	✗	✓

6.4.3 Success Evaluation

This evaluation was conducted with two main purposes in mind. The first one was to determine if the right method is built and supports achieving the aim of the research. Secondly, it was important to know whether the proposed approach has the expected impact on the success factor i.e. architectural design space exploration (ADSE), and is indeed useful to the practicing architects.

The 'initial' Descriptive Study (II) (see research methodology in Chapter 1) was conducted to evaluate the research by experts from industry. To this purpose, an evaluation session was organised with experts from a well-known aircraft manufacturer. Five experts attended the session. The questionnaire used is given in Appendix F. The outline of the session is shown in Table 6.10. At the start, the author introduced the research philosophy in the form of a presentation. All the questions raised during the presentation were answered. Furthermore, it was ensured that the participants sufficiently comprehend the proposed approach. Then, the approach was demonstrated on a hypothetical transport aircraft design scenario in Section 6.3. Then, some time was given to the experts to ask questions related to the demonstrated prototype as well as to the research, in general. Next, the questionnaire was filled in by the experts. The open-ended questions of the questionnaire were discussed by the experts in detail. The participants' years of experience are presented in Table 6.11.

TABLE 6.10: Outline of the evaluation session.

Sr. No	Activity	Duration	Involvement
1	Introductory presentation	20 - 25 min	Author
2	Demonstration of the approach on the prototype tool	10 - 15 min	Author
3	Questions and discussion	30 - 40 min	Author and experts
4	Questionnaire	25 - 30 min	Experts

TABLE 6.11: Experts involved in the evaluation.

Participant	Experience in years
P1	10 +
P2	30
P3	18
P4	30
P5	10 +

The questionnaire contained two types of questions. The initial four questions are of 'Likert' type, where possible options are provided for each of the questions and the most appropriate option is asked to be selected within the context of the research. The remaining three are 'Open-ended' questions. Here, the questions are expected to be answered in detail. The feedback given by the experts is summarised below.

The feedback on each of the 'Likert' type of questions is represented in the form of tables and plot, and followed by a description of the results, as follows.

TABLE 6.12: Q. 1 (a).

Q. 1 (a)	Options	P1	P2	P3	P4	P5
The proposed method would enhance interactivity during the Architecture Definition (AD) and Architecture Assessment (AA) processes.	Strongly disagree					
	Disagree					
	Neither agree or disagree					
	Agree		✓	✓		✓
	Strongly agree	✓			✓	

From the feedback provided on the question 1(a) of the questionnaire, it can be seen, as shown in Table 6.12 and Figure 6.22, that all the participants agree that the proposed method enhances the interactivity during the architecture definition (AD) and assessment (AA) processes.

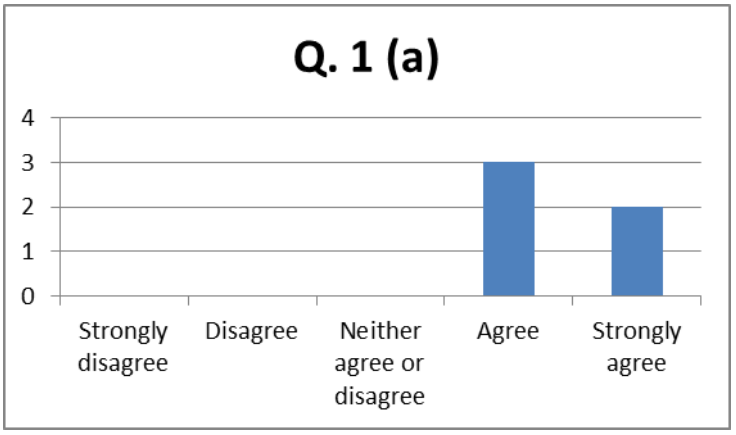


Figure 6.22: *Assessment of the feedback on Q. 1 (a).*

TABLE 6.13: Q. 1 (b).

Q. 1 (b)	Options	P1	P2	P3	P4	P5
The proposed method is likely to enable me to reduce the number of time-consuming manual activities in AD and AA.	Strongly disagree					
	Disagree					
	Neither agree or disagree					✓
	Agree		✓	✓	✓	
	Strongly agree	✓				

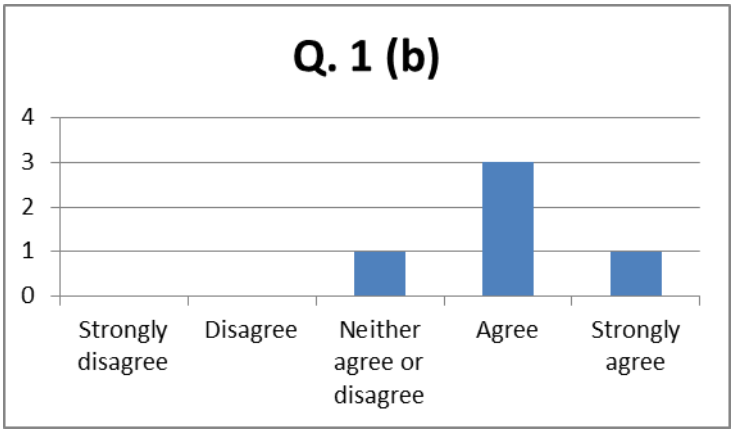


Figure 6.23: *Assessment of the feedback on Q. 1 (b).*

Q. 1 (b) is about the method's ability to reduce the number of time-consuming manual activities in AD and AA. No one disagreed (see Table 6.13 and Figure 6.23); however, one of the participants was neutral on this question.

TABLE 6.14: Q. 1 (c).

Q. 1 (c)	Options	P1	P2	P3	P4	P5
It is reasonable to assume that component computational models would be available during conceptual design.	Strongly disagree					
	Disagree					
	Neither agree or disagree					✓
	Agree		✓	✓	✓	
	Strongly agree	✓				

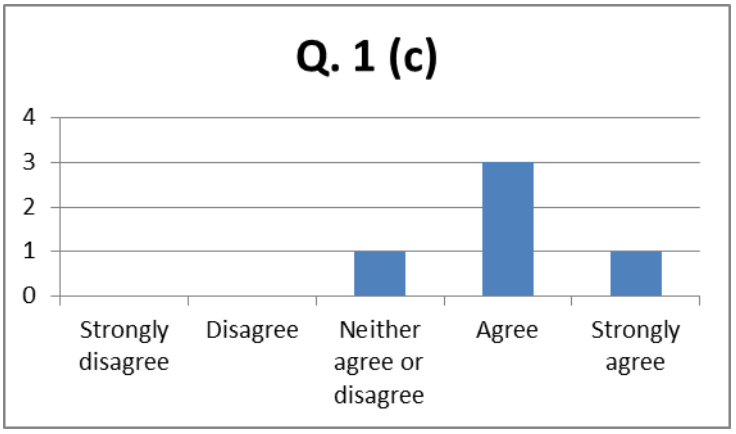


Figure 6.24: *Assessment of the feedback on Q. 1 (c).*

The feedback given by the participants supports the assumption that component computational models would be available during conceptual design, as shown in Table 6.14 and Figure 6.24. All participants agree that the assumption is reasonable except one who preferred to be neutral and provided the reason of not having sufficient experience in this regard.

All the participants appreciated the practicability of the proposed approach and selected the option, either 'good' or 'excellent', as summarised in Table 6.15 and Figure 6.25. Participant, P2, missed answering this question.

Summarised below is the feedback given on the 'open-ended' questions. The detailed comments given by the experts are given in Appendix F.

TABLE 6.15: Q. 2.

Q. 2	Options	P1	P2	P3	P4	P5
Practicality of the proposed method.	Poor					
	Fair					
	Average					
	Good			✓	✓	✓
	Excellent	✓				

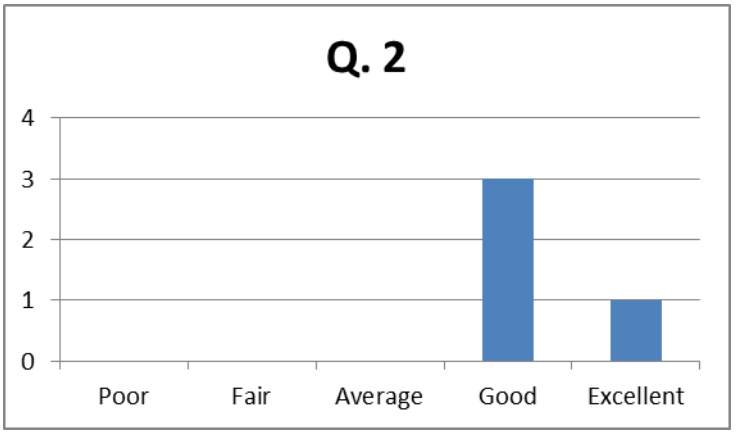


Figure 6.25: *Assessment of the feedback on Q. 2.*

Overall, the analysis of the 'open-ended' questions confirmed that the usability of the proposed approach. The method's capabilities and advantages praised by the experts are as follows:

- Ease of use
- Ability to rapidly evaluate architectures
- Rapid adaptation to architectural changes with minimum 'setting effort'
- 'Human in loop' during design
- Re-use of workflows

The participants also pointed out areas for further improvements beyond the scope of the current research as below:

- Capture design rationale

- Consider uncertainty on outputs and constraints
- Browsing/searching through database of the architectures
- Consider failure cases during sizing
- Consideration of physical view and mission analysis

6.5 Summary and Conclusions

Presented in this chapter are the results of the evaluations conducted for the research.

The object-oriented approach was employed to design and implement an evaluation prototype tool which was employed to demonstrate and evaluate the proposed methods. Three types of evaluations were required to be conducted for the kind of research presented in this thesis.

The first type, 'success' evaluation, was conducted to verify the correctness of the developed methods, i.e., assessment of consistency and in-built functionality of the methods. To this purpose, the results of the application of the method on representative test-cases were compared to existing method wherever possible. The comparison confirmed the correctness of the methods for inbuilt functionality and consistency.

The second type, 'application' evaluation, was performed to confirm that the application of the method has the intended effect on the 'key' factors of the impact model. That is, the application of the method reduces the involved manual tasks in both architecture definition and assessment. It was found that the method indeed reduces the involved manual tasks by automating them.

The third type, 'success' evaluation, was conducted to confirm the usefulness of the method in industry. To this purpose, a hypothetical aircraft design scenario was demonstrated to experts from the aircraft industry. Their feedback confirmed the usefulness of the method in the industrial setup. Furthermore, it was also confirmed by the experts that the method indeed accelerates the architectural design space exploration (ADSE) and makes it an interactive design activity. The experts also provided some suggestions

on how to develop the method further. The suggestions will be addressed as a part of future work in Chapter 7.

CHAPTER 7

Conclusions and Future Work

This chapter provides a summary of the work undertaken. It includes an overview of the research, and a discussion on its primary contributions. Also, avenues for the future research are outlined.

7.1 Introduction

The aim of the research was to develop a methodology that increases the efficiency of the conceptual stage architectural design space exploration while making it a more interactive process. This aim has been met through the development of several enablers and a prototype architecting tool which support the architect in the rapid and interactive definition and assessment of airframe architectures.

In this chapter, Section 7.2 provides a summary of the research. The main contributions to knowledge are presented in section 7.3. Finally, the limitations of the current research and avenues of the future work are discussed in Section 7.4.

7.2 Summary of research

7.2.1 Literature review

A literature survey was conducted. It identified a number of existing state of the art methods for conceptual design, including architecting tools/languages, *functional reasoning*, sizing and analysis, and construction of computational workflows.

One of the industrial requirements from the TOICA project[13] for the architecting process was a traceability mechanism that enables the architect to find relations among architectural elements. The existing methods do not support dependency analysis. The available languages used for architecting are complex to use and lack interactivity during the architecting process. The review also identified a large number of *functional reasoning* models. These models have different definitions of a 'function'. Therefore, the knowledge stored cannot be shared with other reasoning models. Furthermore, the *functional reasoning* models focused on only functional and logical views of the system architecture.

Various methods, software tools, and languages relating to architectural design and analysis were also reviewed to identify their potential to be applied in the current research context. It was found that there exist methods which do automate some of the activities of the architecture sizing and analysis. It appears that the sizing workflows are assumed to be available and given the required information (for sizing), are called to compute the sizes. This may not always be the case, because depending on the user input, the workflow is required to be re-configured. During this process, it is possible that the system of models could be under-determined or over-determined. Available tools do not handle such situations.

The review of the methods related to workflow management found that most of the methods deal with single output models/equations. The only method which handles models with multiple inputs and outputs requires a default determined workflow as input. It was also found that constraint management methods had the potential to be used in the research with some modifications for the current research needs.

7.2.2 Methods for supporting architecture definition and assessment process

A novel method comprised of mainly two parts was introduced to support conceptual design tasks, in particular, architecture definition, and sizing and performance evaluation. The first part focused on supporting the architecting process, whereas, the second part focused on supporting the sizing and performance evaluation of the system architecture. The purpose of these methods was to reduce tedious time-consuming manual activities, and they are expected to improve the efficiency of the architectural design space exploration (ADSE).

A method that enables the dependency analysis of the architectural elements from different views of the system architecture has been proposed. Here, the mapping/fulfilment associations between the elements of the architecture are modelled as 'relations' and captured in the form of a graph. The associations made by the user are referred to as 'direct relations', whereas the 'indirect relations' which are a consequence of the 'direct relations' are produced behind the scenes using 'Transitive Closure' algorithm. Furthermore, in addition to this, a separate data structure, a bipartite graph, is employed for capturing *functional reasoning* knowledge. This only contains elements of functional and logical views of the system architecture and relations among them. Both of these graphs are used for enabling the inter domain traceability in combination with a depth-first search (DFS) algorithm. Also, the sizing workflows together with DFS algorithm are used to trace the dependency within the computational view.

To support architecture assessment i.e. the architecture sizing and performance evaluation, a three-step method was proposed. It takes the logical view of the architecture and steady-state models behind each of the logical components as input. In the first step, the sequence of the sub-systems is automatically generated by extracting a sub-systems source-sink 'Dependency Structure Matrix (DSM)' from the logical view, followed by the application of an algorithm which determines the systems' sizing sequence. In the second step, the individual sub-systems and system level workflows are constructed. Here, the computational workflow (a network of computational models) is represented

as a bipartite graph. A maximum matching enumeration algorithm is used to find all possible workflows for a given model set, and another algorithm, to choose from these the most computationally efficient one, i.e., the workflow with the lowest number of reversed variables. In the third step, the workflows produced in the second step and sub-systems' sizing sequence obtained in the first step are combined to produce a complete workflow.

The proposed methods were implemented in a prototype object-oriented architecting tool for demonstration and evaluation. This is summarised in the next section.

7.2.3 Prototype software

For demonstration and evaluation of the various methods devised in this research, the author developed a prototype object-oriented architecting tool, and integrated it with the AirCADia Architect[44], an in-house architecting tool developed during the TOICA* project to create only functional and logical views of the architecture. The integrated-tool supports the definition and assessment of architectures.

It was assumed that the architecting process takes place in the 'Requirement', 'Function', and 'Logical' domains. Different views are proposed to describe the (architecture) system in these domains. Northwood Software's Diagramming library was used to graphically show the information contained in these views. In addition, 'project explorer' window is provided to display all the architectures and their associated common data including '*functional reasoning*' knowledge, requirements, functions etc. A 'Drag-and-drop' mechanism is provided which allows the user to rapidly and interactively create new architectures, using functions and solutions from the library. For traceability purposes, two types of graphs are generated behind the scenes during architecting - one for capturing 'inter domain' relations between the elements and the other one to capture *functional reasoning* knowledge (in the 'Functional' and 'Logical' domains) of the architecture. These graphs are employed to enable dependency analysis of the architectural elements.

*Thermal Overall Integrated Conception of Aircraft

A separate view is provided for sizing and assessing system level performance of the architectures. The view contains four sub-windows for 1) listing individual sub-systems and system workflows, 2) complete workflows, 3) displaying the complete workflow, and 4) showing execution results of a complete workflow.

The tool was used in all the types of evaluations conducted during the research.

7.2.4 Evaluation of the research

Three types of evaluations were conducted. The first type, the support evaluation, focused on checking the consistency and in-built functionality of the developed methods and to confirm that the method is rightly built. Application of the method (using the prototype tool) on various representative test-cases and comparison of the results from the existing methods, on the same test-cases or from manually performing the corresponding tasks, confirmed the correctness of the proposed methods.

The application evaluation was conducted to check if the application of the proposed methods had the intended effect for which the methods were devised. For this, a representative test-case focusing on Environmental Control System (ECS) design was developed. The test-case was recreated in proposed approach using the prototype tool. The functionality provided by the approach during the ECS architecture definition and sizing processes is compared with functionality provided by existing SysML based architecting tool and OpenModelica for architecture definition and sizing processes, respectively. It was found that the method indeed reduces the involved manual tasks compared to the above two approaches.

Finally, the third type of evaluation (success evaluation) was conducted to determine if the methods are indeed useful in the actual industrial setup. For this, the developed methods were demonstrated to experts from industry and feedback was sought. The evaluation session involved the author introducing the research and philosophy behind it, followed by a demonstration of the methods using AirCADia Architect on a hypothetical design scenario of a transport aircraft design, and discussion and feedback (in the form of a questionnaire). The feedback from the experts confirmed the usefulness of

the method. Furthermore, the experts provided suggestions for improving the method. These suggestions will be addressed in Section 7.4.

The aim of the research was to improve the efficiency of the architectural design space exploration. These evaluations confirmed that application of the developed methods /support improves the existing situation (as described in the impact model shown in Figure 3.23 and 4.1) by increasing automation in both architecture definition and assessment, thus, improves the efficiency of the exploration.

7.3 Novelty and Contribution to knowledge

The main contributions to knowledge of the presented research are summarised as follows:

- Integrated computational framework that allows the user to interactively and rapidly define new system architectures and assess their performance at the system level.
- A novel method for supporting architecture definition tasks which allows:
  - Architects to perform dependency analysis of architectural elements within and between different views of the architecture.
  - Capturing of *functional reasoning* knowledge and interactive mechanism to employ the existing *functional reasoning* knowledge in the new architectures.
- A novel method for supporting architecture sizing and performance evaluation, including algorithms for:
  - Automated sequencing of the sub-systems for sizing elements within and between different views of the architecture.

  - Automated construction of computational sizing workflows for individual sub-systems and at system level. This also involves supporting or/and assisting user in situations where the system of models is under-determined and over-determined.
  - Automated construction of a complete workflow, combining the sequence and workflows.

7.4 Current limitations and avenues of future work

This section describes the limitations of the proposed approach and avenues for future work that could address these limitations as follows:

- Currently, the architecture is sized for a given point in the system mission, specifically, cruise. However, it is necessary to determine the worst case scenario for a sub-system to be sized, given the system mission. It is important to investigate how the whole mission can be incorporated in the proposed approach.
- Early consideration of the physical sizes of the sub-systems and overall layout of the associated components (in the system geometry) is necessary to avoid unnecessary design changes in later stages, e.g. due to packaging (spatial interference) issues. Also, consideration of the physical view allows further investigation, such as thermal analysis.
- The proposed approach for aggregating variables up in the hierarchy is limited to additive variables. Currently, variables which are not 'additive' are manually aggregated at a system level. Future work will involve investigation of automated aggregation of the non-additive variables.
- During the success evaluation of the research, the industrial experts suggested to consider capturing design rationale during conceptual stage. They also wished to have a mechanism that could allow them to trace the evolution links of the architecture to its ancestor architectures and browse/search through the database

of previous architectures. Unfortunately, this is beyond the scope of the research. However, it is important to address the needs of practicing engineers, and will need further research to address the above challenges.

- During conceptual design very little is known about the system to be designed, but more design freedom is available. Therefore, decisions are required to be taken with little available design information, and with appropriate assumptions. Furthermore, the computational models employed for sizing the architecture may not be completely accurate. This research has not been primarily concerned with 'uncertainty'. However, it is vital to consider 'uncertainty' during the design process to make design decisions with sufficient confidence.

- This research did not concentrate on development of methods for solving the sizing workflows. However, it has been attempted to reduce feedback loops while solving strongly connected components (SCCs) contained in the workflow, and this has been successfully tested only on the test-case which the author developed. This is required to be tested further thoroughly. Furthermore, it was found that the gradient based numerical methods or fixed point iteration (FPI) for solving modified models or SCCs are not sufficiently robust to find the solution as their convergence greatly depends upon the starting values given for guess variables during iterative solving. More investigation is necessary to explore the results, where multiple solutions exist. This area was beyond the scope of this work.

- During the success evaluation, the experts also suggested considering failure modes while sizing the sub-system, and failure mode effects analysis (FMEA) during early stages of the design. The author does think that the method could be adapted for different failure cases during the sizing. However, to corroborate the previous statement, further research is needed.

References

[1] G. Pahl and W. Beitz, *Engineering design: a systematic approach*. Springer Science & Business Media, 2013.

[2] J. Jänsch and H. Birkhofer, "The development of the guideline VDI 2221-the change of direction," in *DS 36: Proceedings DESIGN 2006, the 9th International Design Conference, Dubrovnik, Croatia*, 2006.

[3] M. Tooley, *Design engineering manual*. Elsevier, 2009.

[4] P. Freeman and A. Newell, "A Model for functional reasoning in design," in *Proceedings of the 2Nd International Joint Conference on Artificial Intelligence*, IJCAI'71, (San Francisco, CA, USA), pp. 621–640, Morgan Kaufmann Publishers Inc., 1971.

[5] A. Chakrabarti and T. P. Bligh, "A scheme for functional reasoning in conceptual design," *Design Studies*, vol. 22, no. 6, pp. 493–517, 2001.

[6] Y. Umeda and T. Tomiyama, "Functional reasoning in design," *IEEE expert*, vol. 12, no. 2, pp. 42–48, 1997.

[7] A. Kossiakoff, W. N. Sweet, S. J. Seymour, and S. M. Biemer, *Systems Engineering Principles and Practice*, vol. 83. John Wiley & Sons, 2011.

[8] I. Chakraborty, D. N. Mavris, M. Emeneth, and A. Schneegans, "An integrated approach to vehicle and subsystem sizing and analysis for novel subsystem architectures," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 230, no. 3, pp. 496–514, 2016.

[9] I. Chakraborty, D. N. Mavris, M. Emeneth, and A. Schneegans, "A system and mission level analysis of electrically actuated flight control surfaces using Pacelab SysArc," in *52nd Aerospace Sciences Meeting*, (National Harbor, Maryland), p. 381, 2014.

[10] C. Ingram, T. Dendinger, E. Inclan, Y. Charront, K. Handschuh, I. Chakraborty, E. Garcia, and D. N. Mavris, "Integrating subsystem sizing into the more electric aircraft conceptual design phase," in *53rd AIAA Aerospace Sciences Meeting*, p. 1682, 2015.

[11] I. Chakraborty, D. R. Trawick, D. N. Mavris, M. Emeneth, and A. Schneegans, "A requirements-driven methodology for integrating subsystem architecture sizing and analysis into the conceptual aircraft design phase," in *14th AIAA Aviation Technology, Integration, and Operations Conference*, p. 3012, 2014.

[12] M. Saravi, L. Newnes, A. R. Mileham, and Y. M. Goh, "Estimating cost at the conceptual design stage to optimize design in terms of performance and cost," in *Collaborative product and service life cycle management for a sustainable world*, pp. 123–130, Springer, 2008.

[13] "TOICA-FP7 - Thermal Overall Integrated Conception of Aircraft." Available at www.toica-fp7.eu/,(Accessed: 6 April 2018).

[14] L. T. M. Blessing and A. Chakrabarti, *DRM, a design research methodology*. Dordrecht; New York: Springer, 2009.

[15] "What is Systems Engineering." Available at https://www.incose.org/AboutSE/WhatIsSE, (Accessed: 6 April 2018).

[16] "Welcome to Cranfield University." Available at https://www.cranfield.ac.uk/, (Accessed: 6 April 2018).

[17] V. Datta, *Interactive computational model-based design: a blackbox perspective*. Phd thesis, School of Engineering, Cranfield University, 2014.

[18] L. K. Balachandran, *Computational workflow management for conceptual design of complex systems: an air-vehicle design perspective*. PhD thesis, School of Engineering, Cranfield University, 2007.

[19] H. Yoshikawa and K. Uehara, "Design theory for cad/cam integration," *CIRP Annals-Manufacturing Technology*, vol. 34, no. 1, pp. 173–178, 1985.

[20] Y. Umeda, T. Tomiyama, and H. Yoshikawa, "FBS modeling: modeling scheme of function for conceptual design," in *Proceedings of the 9th international workshop on qualitative reasoning*, pp. 271–278, 1995.

[21] B. Hamraz and P. J. Clarkson, "Industrial evaluation of FBS Linkage–a method to support engineering change management," *Journal of Engineering Design*, vol. 26, no. 1-3, pp. 24–47, 2015.

[22] B. Hamraz, N. H. M. Caldwell, D. C. Wynn, and P. J. Clarkson, "Requirements-based development of an improved engineering change management method," *Journal of Engineering Design*, vol. 24, no. 11, pp. 765–793, 2013.

[23] R. B. Stone and K. L. Wood, "Development of a Functional Basis for Design," *Journal of Mechanical design*, vol. 122, no. 4, pp. 359–370, 2000.

[24] J. M. Hirtz, R. B. Stone, S. Szykman, D. A. McAdams, and K. L. Wood, "Evolving a functional basis for engineering design," in *Proceedings of the ASME Design Engineering Technical Conference: DETC2001, Pittsburgh, PA*, 2001.

[25] J. Zou and Q. Du, "A functional reasoning cube model for conceptual design of mechatronic systems," *Strojniški vestnik-Journal of Mechanical Engineering*, vol. 59, no. 5, pp. 323–332, 2013.

[26] N. P. Suh, "Axiomatic design: advances and applications (The Oxford Series on Advanced Manufacturing)," 2001.

[27] "Function Analysis System Technique (FAST) - Canadian Society of Value Analysis." Available at www.valueanalysis.ca/fast.php, (Accessed: 6 April 2018).

[28] J. R. Wixson, "Function analysis and decomposistion using function analysis systems technique," in *INCOSE International Symposium*, vol. 9, pp. 800–805, Wiley Online Library, 1999.

[29] J. Borza, "Fast diagrams: The foundation for creating effective function models," *trizcon 2011, Detroit*, 2011.

[30] C. W. Bytheway, *FAST creativity and innovation: Rapidly improving processes, product development and solving complex problems*. J. Ross Publishing, 2007.

[31] D. D. E. Tate, *A roadmap for decomposition: activities, theories, and tools for system design*. PhD thesis, Massachusetts Institute of Technology, 1999.

[32] D. van Eck, D. A. McAdams, and P. E. Vermaas, "Functional decomposition in engineering: a survey," in *ASME 2007 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pp. 227–236, American Society of Mechanical Engineers, 2007.

[33] L. Delligatti, *SysML distilled: A brief guide to the systems modeling language*. Addison-Wesley, 2013.

[34] T. Weilkiens, *Systems engineering with SysML/UML: modeling, analysis, design*. Elsevier, 2011.

[35] "OMG — Object Management Group." Available at www.omg.org/,(Accessed: 6 April 2018).

[36] "Welcome To UML Web Site!." Available at http://www.uml.org/,(Accessed: 6 April 2018).

[37] "OMG SysML Home — OMG Systems Modeling Language." Available at http://www.omgsysml.org/, (Accessed: 6 April 2018).

[38] C. J. J. Paredis, Y. Bernard, R. M. Burkhart, H.-P. Koning, S. Friedenthal, P. Fritzson, N. F. Rouquette, and W. Schamai, "An overview of the SysML-Modelica transformation specification," in *INCOSE International Symposium*, vol. 20, pp. 709–722, Wiley Online Library, 2010.

[39] M. Tiller, *Introduction to Physical Modeling with Modelica*. Norwell, MA, USA: Kluwer Academic Publishers, 2001.

[40] P. Fritzson, *Principles of object-oriented modeling and simulation with Modelica 3.3: a cyber-physical approach*. John Wiley & Sons, 2014.

[41] SAE, "Architecture analysis and design language (aadl)," *AS5506C Standard, SAE International*, 2017.

[42] J. Delange, *AADL In Practice: Become an expert in software architecture modeling and analysis*. Reblochon Development Company, 2017.

[43] D. P. Gluch and P. H. Feiler, *AADL In Practice: Become an expert in software architecture modeling and analysis*. Pearson Education (US), 2012.

[44] M. Guenov, A. Molina-Cristóbal, V. Voloshin, A. Riaz, A. S. van Heerden, S. Sharma, C. Cuiller, and T. Giese, "Aircraft systems architecting a functional-logical domain perspective," in *16th AIAA Aviation Technology, Integration, and Operations Conference*, (Washington, D.C.), p. 3143, 2016.

[45] S. Kleiner and C. Kramer, "Model based design with systems engineering based on RFLP using V6," in *Smart Product Engineering: Proceedings of the 23rd CIRP Design Conference*, pp. 93–102, Bochum, Germany: Springer, 2013.

[46] "NASA Software." Available at https://software.nasa.gov/software/LAR-18934-1, (Accessed: 6 April 2018).

[47] S. Liscouët-Hanke, J. C. Mare, and S. Pufe, "Simulation framework for aircraft power system architecting," *Journal of Aircraft*, vol. 46, no. 4, pp. 1375–1380, 2009.

[48] I. Chakraborty, *Subsystem architecture sizing and analysis for aircraft conceptual design*. PhD thesis, Georgia Institute of Technology, 2015.

[49] I. Chakraborty, D. N. Mavris, M. Emeneth, and A. Schneegans, "A methodology for vehicle and mission level comparison of more electric aircraft subsystem solutions: Application to the flight control actuation system," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 229, no. 6, pp. 1088–1102, 2015.

[50] D. M. Judt and C. P. Lawson, "Application of an automated aircraft architecture generation and analysis tool to unmanned aerial vehicle subsystem design," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 229, no. 9, pp. 1690–1708, 2015.

[51] D. M. Judt and C. Lawson, "Development of an automated aircraft subsystem architecture generation and analysis tool," *Engineering Computations*, vol. 33, no. 5, pp. 1327–1352, 2016.

[52] C. De Tenorio, D. Mavris, E. Garcia, and M. Armstrong, "Methodology for aircraft system architecture sizing," in *26th international congress of the aeronautical sciences*, vol. 30, 2008.

[53] C. De Tenorio, *Methods for collaborative conceptual design of aircraft power architectures*. Georgia Institute of Technology, 2010.

[54] M. Armstrong, C. De Tenorio, D. Mavris, and E. Garcia, "Function based architecture design space definition and exploration," in *The 26th Congress of ICAS and 8th AIAA ATIO*, p. 8928, 2008.

[55] D. Mavris, C. de Tenorio, and M. Armstrong, "Methodology for aircraft system architecture definition," in *46th AIAA Aerospace Sciences Meeting and Exhibit*, p. 149, 2008.

[56] J. J. Michalek and P. Y. Papalambros, "Weights, norms, and notation in analytical target cascading," *Journal of Mechanical Design*, vol. 127, no. 3, pp. 499–501, 2005.

[57] H. M. Kim, M. Kokkolaras, L. S. Louca, G. J. Delagrammatikas, N. F. Michelena, Z. S. Filipi, P. Y. Papalambros, J. L. Stein, and D. N. Assanis, "Target

cascading in vehicle redesign: a class VI truck study," *International journal of vehicle design*, vol. 29, no. 3, pp. 199–225, 2002.

[58] J. R. Martins and A. B. Lambe, "Multidisciplinary design optimization: a survey of architectures," *AIAA journal*, vol. 51, no. 9, pp. 2049–2075, 2013.

[59] H. M. Kim, D. G. Rideout, P. Y. Papalambros, and J. L. Stein, "Analytical Target Cascading in Automotive Vehicle Design," *Journal of Mechanical Design*, vol. 125, no. 3, p. 481, 2003.

[60] X. Fei, B. German, and S. Parashar, "Implementation of analytical target cascading for aircraft design," in *50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, p. 334, 2012.

[61] B. Helms, K. Shea, and F. Hoisl, "A framework for computational design synthesis based on graph-grammars and function-behavior-structure," in *ASME 2009 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pp. 841–851, American Society of Mechanical Engineers, 2009.

[62] B. Helms and K. Shea, "Computational synthesis of product architectures based on object-oriented graph grammars," *Journal of Mechanical Design*, vol. 134, no. 2, p. 021008, 2012.

[63] C. Münzer, K. Shea, and B. Helms, "Solving design tasks in engineering using object-oriented graph-based representations and boolean satisfiability," in *ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pp. V005T06A016–V005T06A016, American Society of Mechanical Engineers, 2013.

[64] C. Münzer, K. Shea, and B. Helms, "Automated parametric design synthesis using graph grammars and constraint solving," in *ASME 2012 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pp. 517–528, American Society of Mechanical Engineers, 2012.

[65] C. Münzer, B. Helms, and K. Shea, "Automatically transforming object-oriented graph-based representations into boolean satisfiability problems for computational design synthesis," *Journal of Mechanical Design*, vol. 135, no. 10, p. 101001, 2013.

[66] R. Wang and C. H. Dagli, "An executable system architecture approach to discrete events system modeling using SysML in conjunction with Colored Petri Net," in *Systems Conference, 2008 2nd Annual IEEE*, pp. 1–8, IEEE, 2008.

[67] R. Wang and C. H. Dagli, "Executable system architecting using systems modeling language in conjunction with colored petri nets in a model-driven systems development process," *Systems Engineering*, vol. 14, no. 4, pp. 383–409, 2011.

[68] C. H. Dagli, A. Singh, J. P. Dauby, and R. Wang, "Smart systems architecting: computational intelligence applied to trade space exploration and system design," in *Systems Research Forum*, vol. 3, pp. 101–119, World Scientific, 2009.

[69] "Pacelab Suite - PACE." Available at https://www.pace.de/products/pacelab-suite/, (Accessed: 6 April 2018).

[70] "Modelon Steady state: The next big thing?." Available at http://www.modelon.com/entry/steady-state-the-next-big-thing/, (Accessed: 6 April 2018).

[71] L. Balachandran and M. Guenov, "Computational workflow management for conceptual design of complex systems," *Journal of Aircraft*, vol. 47, no. 2, pp. 699–703, 2010.

[72] G. Friedman, *Constraint theory: multidimensional mathematical model management*, vol. 23. Springer Science & Business Media, 2006.

[73] G. J. Friedman and C. T. Leondes, "Constraint theory, part i: fundamentals," *IEEE Transactions on Systems Science and Cybernetics*, vol. 5, no. 1, pp. 48–56, 1969.

[74] G. J. Friedman and C. T. Leondes, "Constraint theory, part ii: model graphs and regular relations," *IEEE transactions on Systems Science and Cybernetics*, vol. 5, no. 2, pp. 132–140, 1969.

[75] D. Serrano, *Constraint management in conceptual design*. Phd thesis, Mechanical Engineering, Massachusetts Institute of Technology, 1987.

[76] H. Yusan and S. Rudolph, "On Systematic knowledge integration in the conceptual design phase of airships," in *13th Lighter-Than-Air Systems Technology Conference*, p. 3909, 1999.

[77] S. Rudolph and M. Bölling, "Constraint-based conceptual design and automated sensitivity analysis for airship concept studies," *Aerospace Science and Technology*, vol. 8, no. 4, pp. 333–345, 2004.

[78] M. Buckley, K. Fertig, and D. Smith, "Design sheet-an environment for facilitating flexible trade studies during conceptual design," in *Aerospace design conference*, (Irvine, California), p. 1191, 1992.

[79] M. W. Bailey and W. H. VerDuin, "Fiper: an intelligent system for the optimal design of highly engineered products," *NIST SPECIAL PUBLICATION SP*, pp. 467–477, 2001.

[80] P. N. Koch, J. P. Evans, and D. Powell, "Interdigitation for effective design space exploration using isight," *Structural and Multidisciplinary Optimization*, vol. 23, no. 2, pp. 111–126, 2002.

[81] K. H. Kwon, W. J. Chung, and K. B. Park, "Application of isight® for optimal tip design of complex tool holder spindle," *ratio*, vol. 850, no. 7.8, pp. 0–3.

[82] "modeFRONTIER web page." Available at http://www.esteco.com/modefrontier, (Accessed: 6 April 2018).

[83] W. Vankan and M. Laban, "A spineware based computational design engine for integrated multi-disciplinary aircraft design," in *9th AIAA/ISSMO symposium on multidisciplinary analysis and optimization*, p. 5445, 2002.

[84] G. Kappel, P. Lang, S. Rausch-Schott, and W. Retschitzegger, "Workflow management based on objects, rules, and roles," vol. 18, pp. 11–18, 01 1995.

[85] M. Rahman, R. Ranjan, R. Buyya, and B. Benatallah, "A taxonomy and survey on autonomic management of applications in grid computing environments," *Concurrency and computation: practice and experience*, vol. 23, no. 16, pp. 1990–2019, 2011.

[86] K. Salimifard and M. Wright, "Petri net-based modelling of workflow systems: An overview," *European journal of operational research*, vol. 134, no. 3, pp. 664–676, 2001.

[87] Y. Bile, A. Riaz, M. D. Guenov, and A. Molina-Cristobal, "Towards automating the sizing process in conceptual (airframe) systems architecting," in *2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, p. 1067, 2018.

[88] T. H. Cormen, *Introduction to algorithms*. MIT press, 2009.

[89] R. Xiao and Q. Fei, "Application of the improved method in structural modeling to comprehensive management of the mine bereaus," *Systems Engineering: Theory and Practive*, vol. 3, no. 17, pp. 57–62 (in Chinese), 1997.

[90] D. Tang, L. Zheng, Z. Li, D. Li, and S. Zhang, "Re-engineering of the design process for concurrent engineering," *Computers & Industrial Engineering*, vol. 38, no. 4, pp. 479–491, 2000.

[91] B. R. Preiss, *Data structures and algorithms with object-oriented design patterns in C++*. John Wiley & Sons, 2008.

[92] A. A. Sonin, "Dimensional analysis," tech. rep., Technical report, Massachusetts Institute of Technology, Technical report, Massachusetts Institute of Technology, 2001.

[93] D. Serrano, "Automatic dimensioning in design for manufacturing," in *Proceedings of the first ACM symposium on Solid modeling foundations and CAD/CAM applications*, pp. 379–386, ACM, 1991.

[94] T. Uno, "Algorithms for enumerating all perfect, maximum and maximal matchings in bipartite graphs," *Algorithms and Computation*, pp. 92–101, 1997.

[95] S. C. Chapra and R. P. Canale, *Numerical methods for engineers: with programming and software applications*. New York, NY, USA: McGraw-Hill, Inc., 3rd ed., 1997.

[96] J. H. Mathews, *Numerical methods for computer science, engineering, and mathematics*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1986.

[97] G. H. Hardy and E. M. Wright, *An introduction to the theory of numbers*. Oxford university press, 1979.

[98] M. Eiglsperger, M. Siebenhaller, and M. Kaufmann, "An efficient implementation of sugiyamas algorithm for layered graph drawing," in *International Symposium on Graph Drawing*, pp. 155–166, Springer, 2004.

[99] "GoDiagram for WinForms - Advanced Controls for .NET Diagrams." Available at https://www.nwoods.com/products/godiagram/, (Accessed: 6 April 2018).

[100] M. A. Dornheim, "Electric cabin: The 787 generates at least four times more electricity than normal. traditionally bleed-powered systems now use volts," *Aviation Week & Space Technology, March*, vol. 28, 2005.

Appendices

APPENDIX A

Publications

Bile, Y., Riaz, A., Guenov, M. D. and Molina-Cristobal, A. [2018], Towards automating the sizing process in conceptual (airframe) systems architecting, in '2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference', p. 1067. **(published)**
(Journal version is in preparation)

Guenov, M.D., Riaz, A., Bile, Y., Molina-Cristobal, A. and van Heerden, A. S., 'Information System Support for Aerospace Vehicle Systems Architecting'.
(submitted to AIAA Journal of Aerospace Information Systems)

Van Heerden, A. S., Guenov, M. D., Molina-Cristobal, A., Riaz, A. and Bile, Y., 'Combined Airframe and Subsystems Evolvability Exploration During Conceptual Design' ICAS 2018 **(Accepted)**

APPENDIX B

Reference and impact model notations

A reference model describes the existing situation in the area of research, whereas, an impact model describes the desired situation in the area. The notations employed in creation of these models are defined here.

B.1 Factor

A factor (or influencing factor) is an aspect of the (existing or desired) situation that influences other aspects of the situation. The factor is expressed as combination of an element and its attribute. The element should be relevant, and measurable and observable. For example, a factor 'quality (*attribute*) of problem definition (*element*)' is shown in Figure B.1. Factors in the reference or impact model are denoted by ellipses.

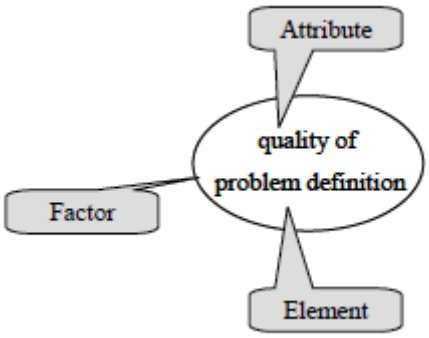


Figure B.1: *Factor, attribute and element*[14].

B.2 Graphical Representation of a Statement

The reference or impact model is a network of factors showing statements about the situation under consideration. Factors are connected by links to form statements. An example of a statement is shown in Figure B.2. The statement requires two factors connected by a link showing values of the attributes at each end. If the link is directed then it is drawn cause to effect factor. Thus, a link describes how the attribute-value of the factor at one end relates to the attribute value at the other end. In the example, the graphical representation states that "a high product quality has a positive effect on customer satisfaction". Each link in the model is labelled with the source(s) of the statement it expresses. The label (source of the statement) is denoted by four different ways for four different sources of the statements.

A number in a square bracket, for instance, $[x]$, conveys that the statement represented by the link is published in reference x. If the statement is assumption then the source is denoted as $[A]$. Whereas, statements based on experience are indicated by $[E]$. Finally, the source of a statement that is based on own investigations is shown by $[O]$. In case of not being known if a link exists between the factors, the source on the link is shown by [?](question mark in the square brackets).

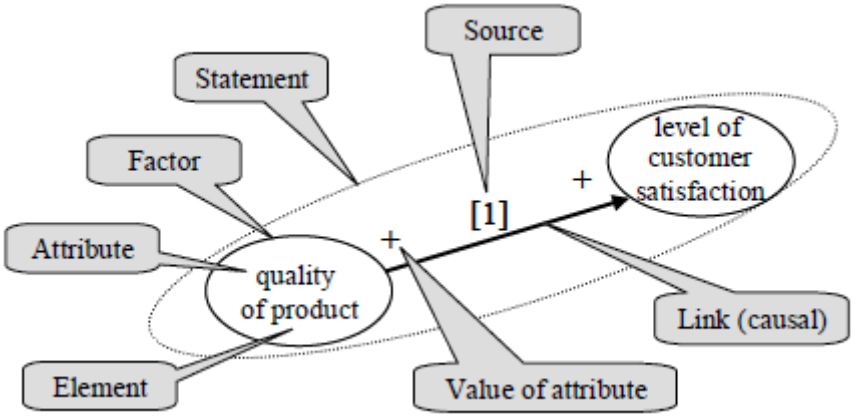


Figure B.2: *Statement representation in a model*[14].

B.2.1 Key Factor

It is an influencing factor (of model) that seems to be the most beneficial factor to address in order to improve an existing situation. In the model, such factors are considered the root cases or the core factors to be addressed to improve the situation. These factors are directly addressed by the support. The support, in the model, is represented by a hexagon.

B.2.2 Success Factor

In the models, success criteria are derived from success factors. These are the factors positioned at the end of the cause-effect network of factors. The preferred values of these factors are taken as the success criteria of the research.

B.2.3 Measurable Success Factor

Due to limited duration of research project, it is impossible to measure the success of the project based on success criteria obtained from success factors; because, the criteria is based on long term effects of the research, such as, increased sales volume, reduced lead-time, improved development process etc., and it is difficult to observe and measure the effect within the time-span of the project. Therefore, it is necessary to choose the

criteria that are obtained from the factors (linked to success factors) from the models which can be measured within the time-frame of the research and is employed to judge the outcomes of the research. Thus, measurable success factors are the factors whose desired values are taken as measurable success criteria (reliable indicators or proxies of the success criteria).

APPENDIX C

Application of Proposed Method on a Test-case

C.1 Modules of Prototype Software Tool

This section describes the different modules of AirCADia Architect which include the requirement, functional, logical and computational views. The main interface of AirCADia Architect is shown in Figure C.1. In this main window, there are sub-windows, including 'Project Explorer', 'Functional Library' and 'Solution Library', 'Welcome' (working area), 'Output', and 'Properties'. The 'Project Explorer' displays the Database and Architecture objects. 'Functional Library' and 'Solution Library' lists all the types of functions and components created by the user. The 'Welcome' window (working area) displays the architecture or database objects and the user can modify them once displayed. The 'Output' shows the status of the execution of the sizing workflows. The 'Properties' window lists down the properties of a selected object in the working area.

Shown in Figure C.2 is an example AirCADia Architect project. Here, it can be seen that the 'Project Explorer' shows the Database objects (Requirements, Zigzagging view,

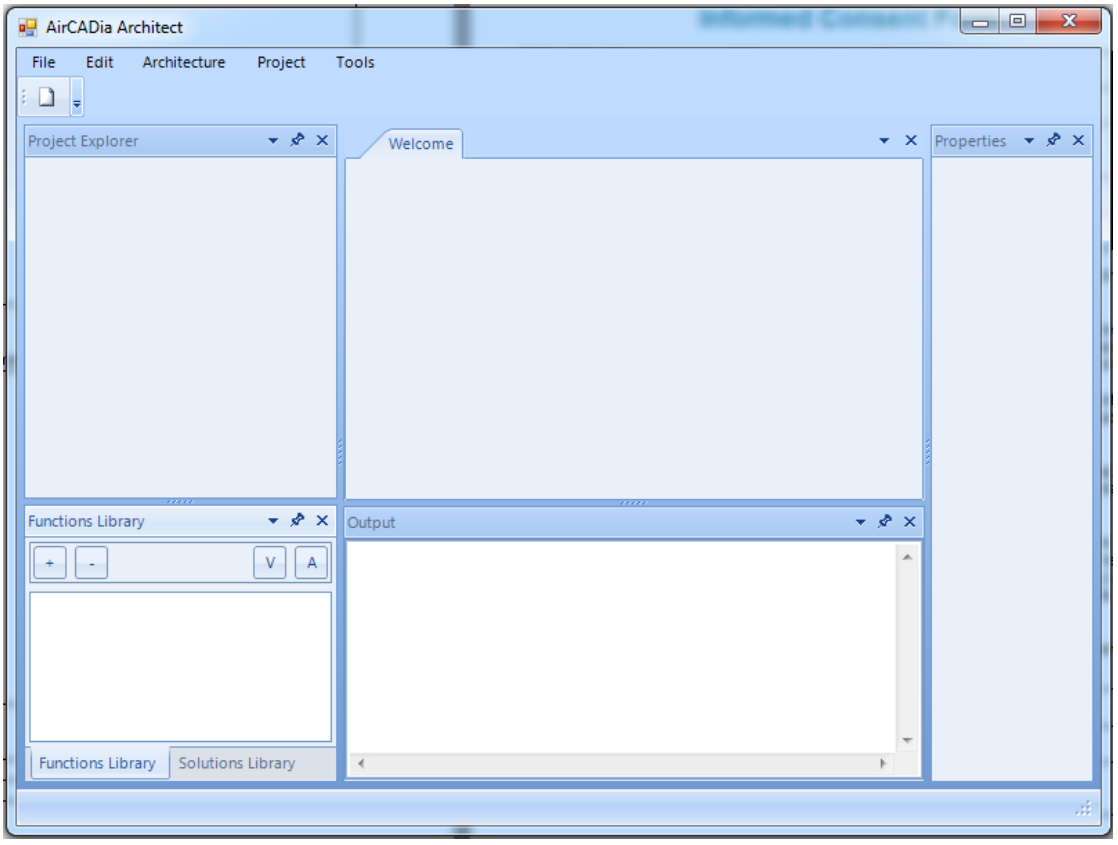


Figure C.1: *Main interface of AirCADia Architect.*

Function-Means Tree and Morphological Matrix) and one Architecture object, containing different views of the architecture. These views contain Requirement View (RV), Functional Hierarchy View (FHV), Functional Flow View (FFV), Logical Hierarchy View (LHV), Logical Flow View (LFV), Functional Logical View (FLV), Computational View (CV), and Scenarios. These views are discussed in detail, later in this section. The 'Functional' and 'Solution' library displays the list of user-defined function- and solution-types.

The main-interface sub-windows can be arranged in the main window by dragging and dropping these windows at a target location on the main window. 'Double-click' on the architecture-views opens the views and the views are displayed on the 'Welcome' window or working area. These views can also be arranged using the 'Drag-drop' mechanism within the working area or on the main window. The architecture views are grouped according to the architecting domains as described below.

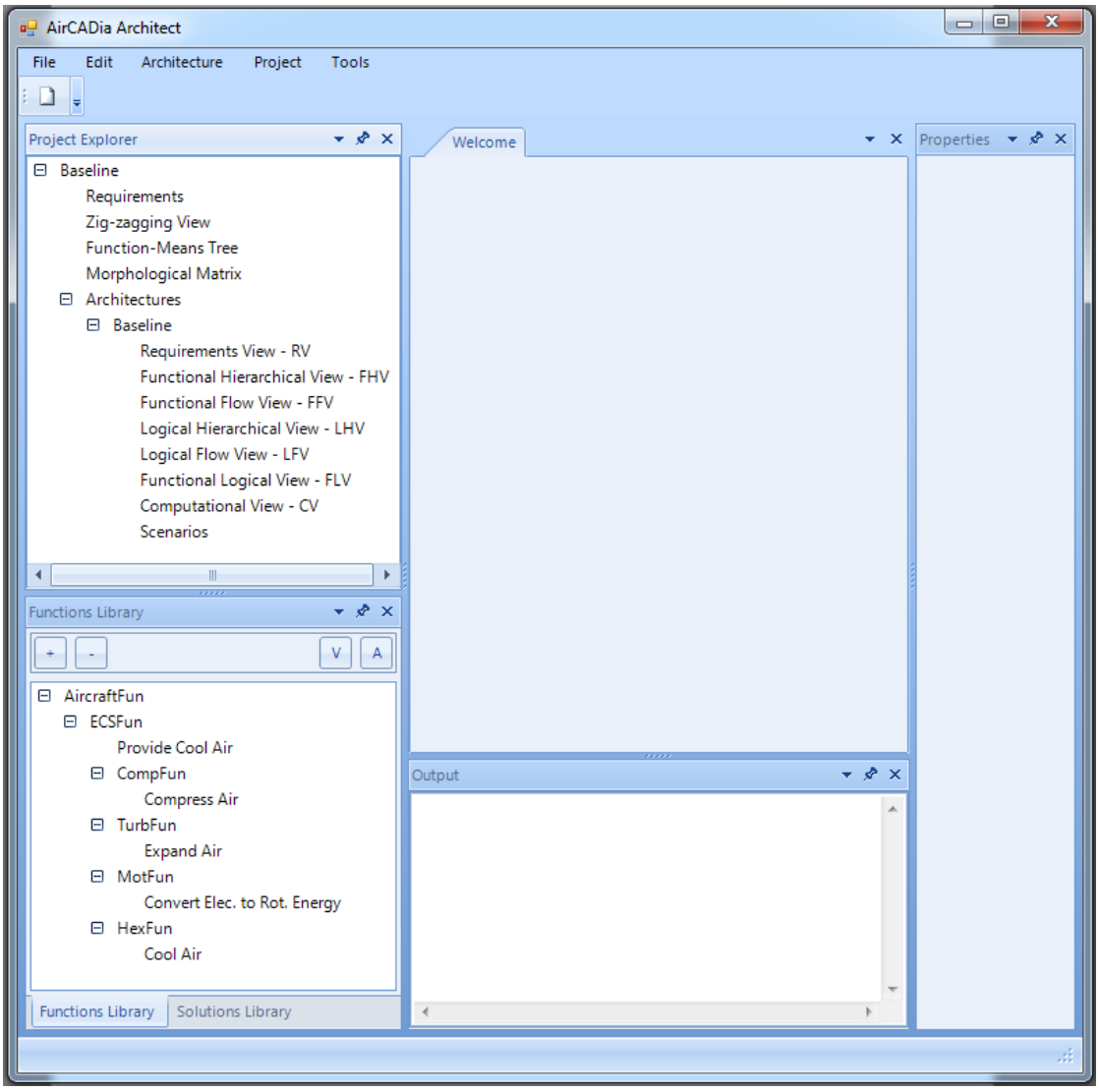


Figure C.2: *An example AirCADia project.*

C.1.1 Requirement Domain

The view of the architecture in requirement domain describes the requirements of the system in a hierarchical form. The view is described by 'Requirement View (RV)' as shown in Figure C.3. This view is provided with two buttons for creating new requirement and mapping requirements to the elements of other views.

After pressing the 'create new requirement' button, a new window pops up as shown in Figure C.4. Here, the user can specify the type of the requirement, i.e. functional or performance, and its description below it. Similarly, while mapping a selected requirement in the RV, different options are shown such as mapping of the requirement

to functions or component. The mapping can be performed interactively i.e. the user just needs to select (through 'mouse' left-click) the requirement and function or component (on the displayed views of the architecture), and click the appropriate option under mapping button. This mapping relationship between the selected architectural elements is captured behind the scene and stored in corresponding elements of the architectural views.

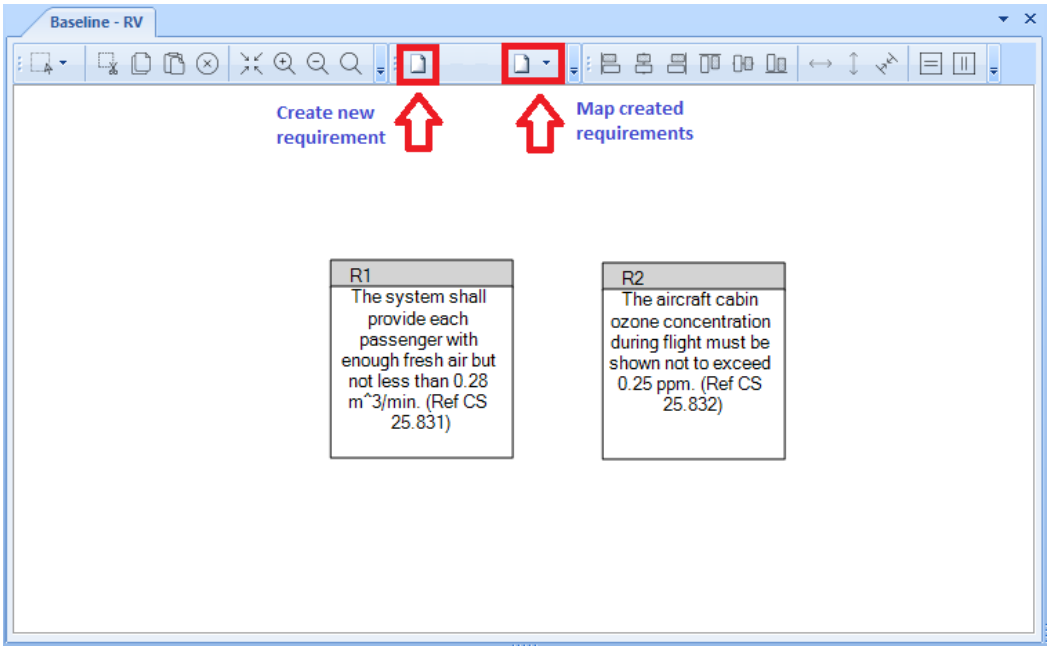


Figure C.3: *Requirement view (RV).*

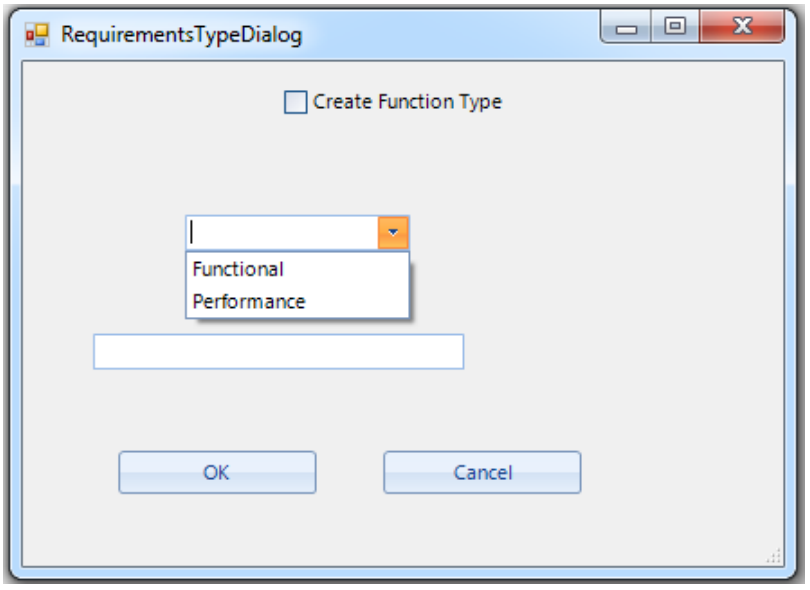


Figure C.4: *New requirement creation window.*

C.1.2 Functional Domain

The architecture description in functional domain contains two views functional hierarchy view (FHV) and functional flow view (FFV) as shown in Figure C.5 and Figure C.6 respectively. The FHV describes the decomposition of the system functions, and FFV describes the flow of information between these functions.

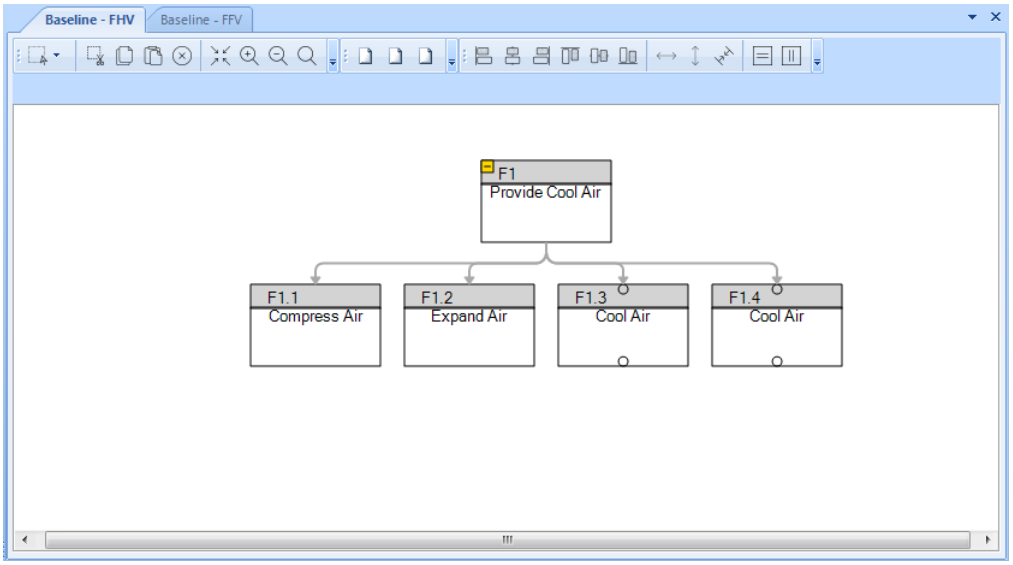


Figure C.5: *Functional hierarchy view (FHV).*

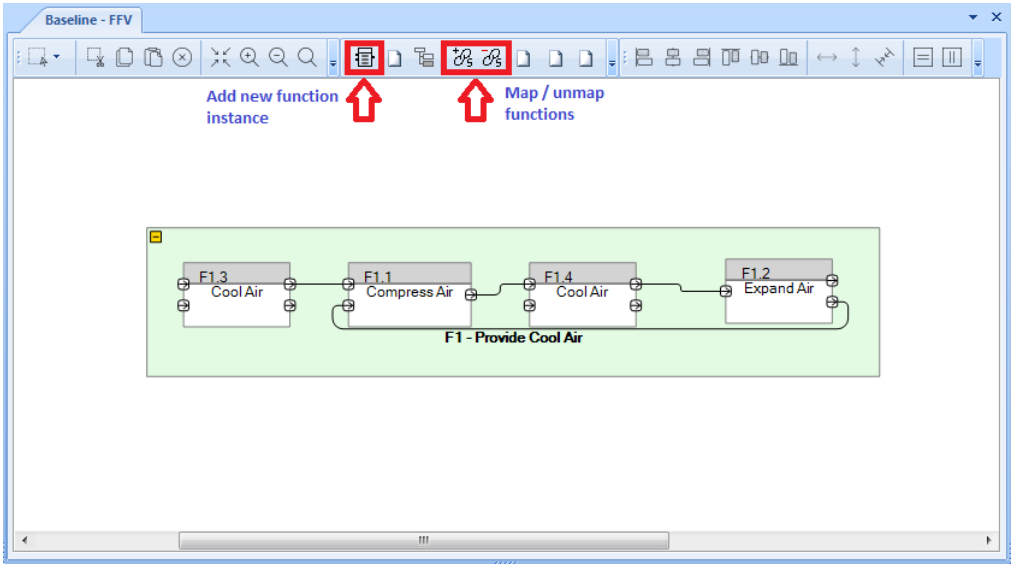


Figure C.6: *Functional flow view (FFV).*

Both the views (FHV and FFV) contain a button which allows the user to add a new instance of the function type from the 'Functional Library'. To do this, the type in the

library is required to be selected and press the button ('Add new function instance' as depicted in Figure C.6) , and a new function instance of the selected function-type gets added in the views. The same can be done by dragging the function type from the library on to the FFV. FHV and FFV are linked to each other, i.e. the addition of the function instance in either of the views (FHV or FFV) automatically adds the function instance in the other view. The user is enabled to find the association between the functions by merely selecting a function in one of these views, and the associated function in the other view gets highlighted. Furthermore, the changes, such as changing the connections in FHV, in the FHV are automatically reflected in the FFV with appropriate modifications behind the scene.

New function types can be added, in the function library. In the functional library window, there is a button for adding the function types in the library. On clicking this button, a new function type creation window shown in Figure C.7 pops up. Here, the function type is defined by specifying the type name and, input and output ports. On clicking the 'OK' button of this window, the created type is added to the function library.

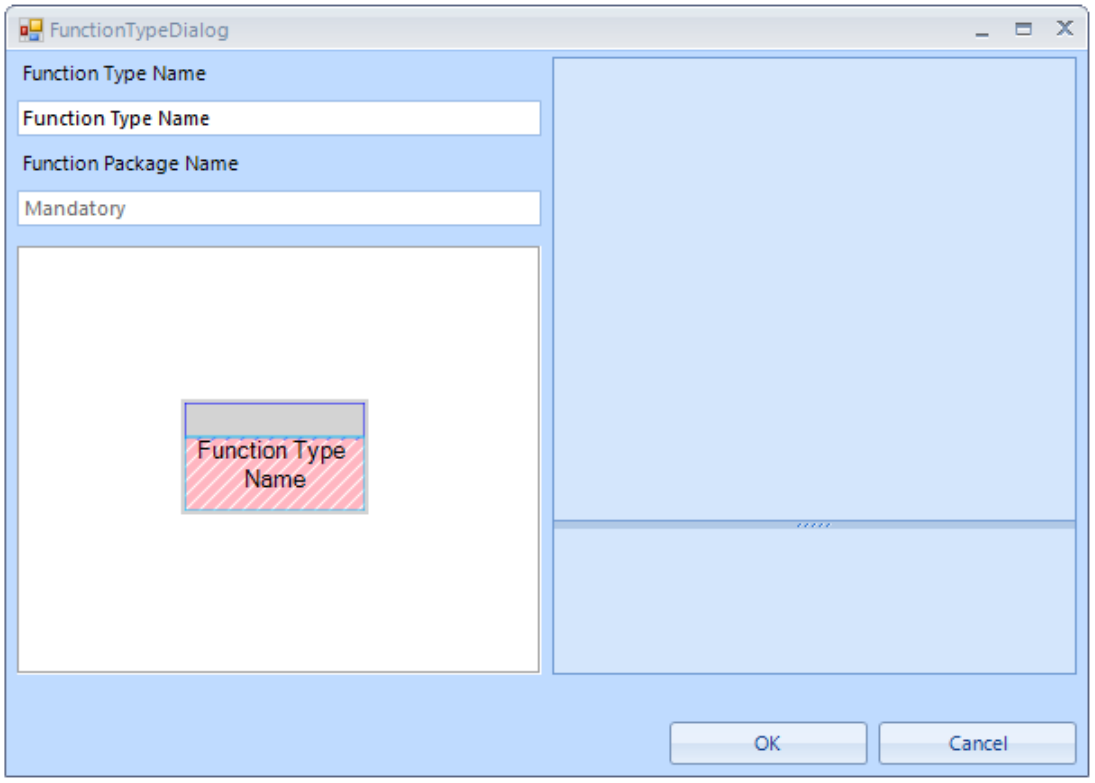


Figure C.7: *New function type creation window.*

C.1.3 Logical Domain

Similar to the functional domain of the system architecture, the logical domain also has two views, logical hierarchy view (LHV) and logical flow view (LFV), as shown in Figure C.8 and Figure C.9, respectively. Similar functionalities as that of FHV and FFV are provided to describe the architecture in LHV and LFV.

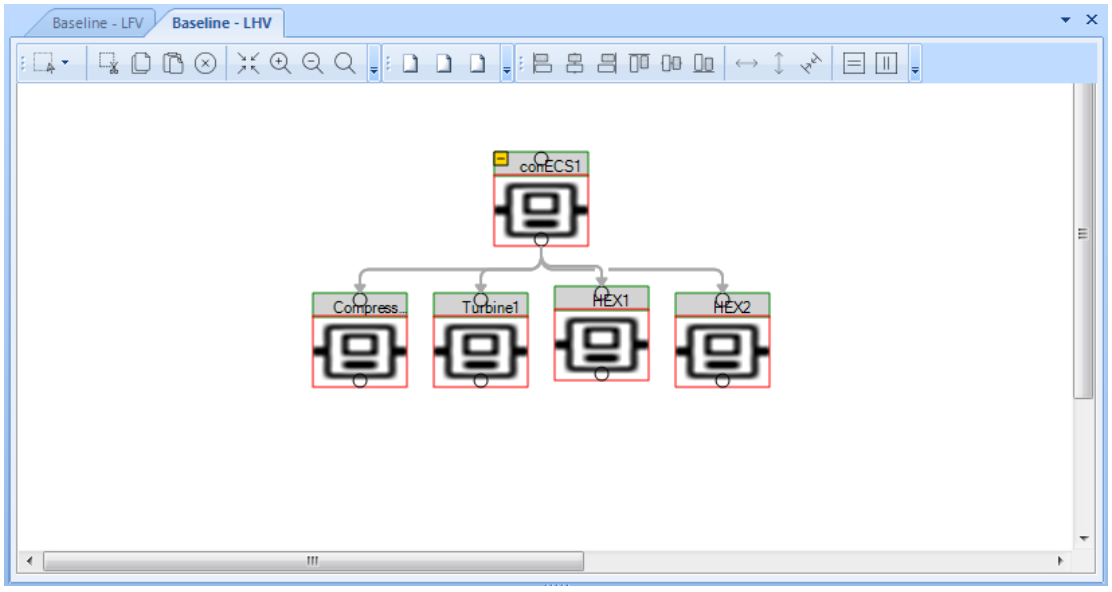


Figure C.8: *Logical hierarchy view (LHV).*

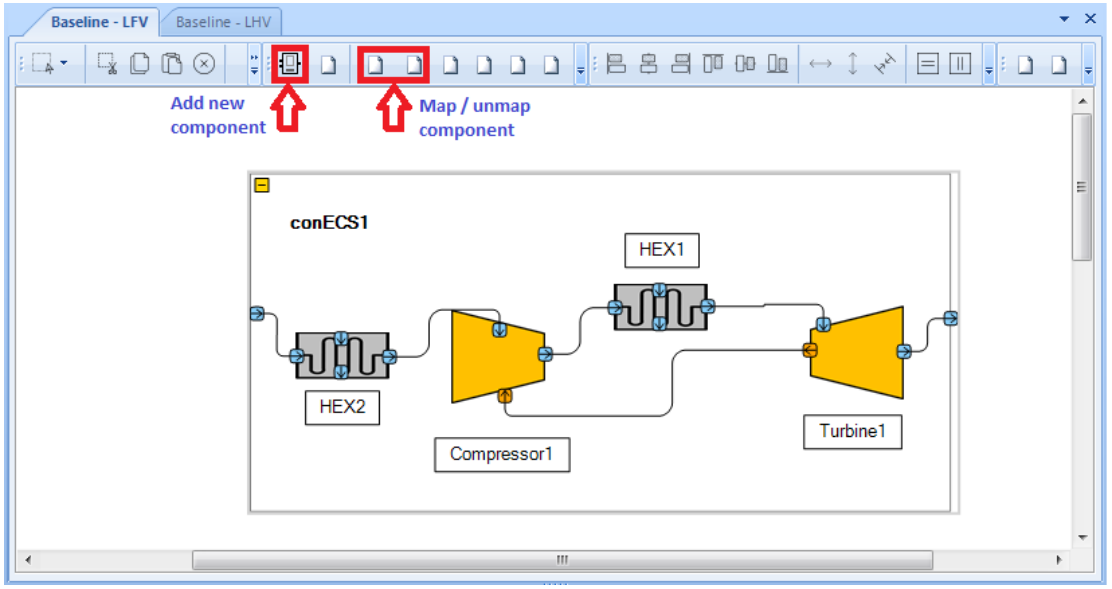


Figure C.9: *Logical flow view (LFV).*

The window for creating a new solution type in the solution library is shown in Figure C.10. Here, the user can customise the appearance of the type and edit the parameters

and computational models associated with the type. After right-click on the type, a context menu appears as shown in Figure C.10. On clicking the 'Edit Parameters' and 'Edit Computational Model' options of the context menu, new windows pop up where parameters and models of the type are edited.

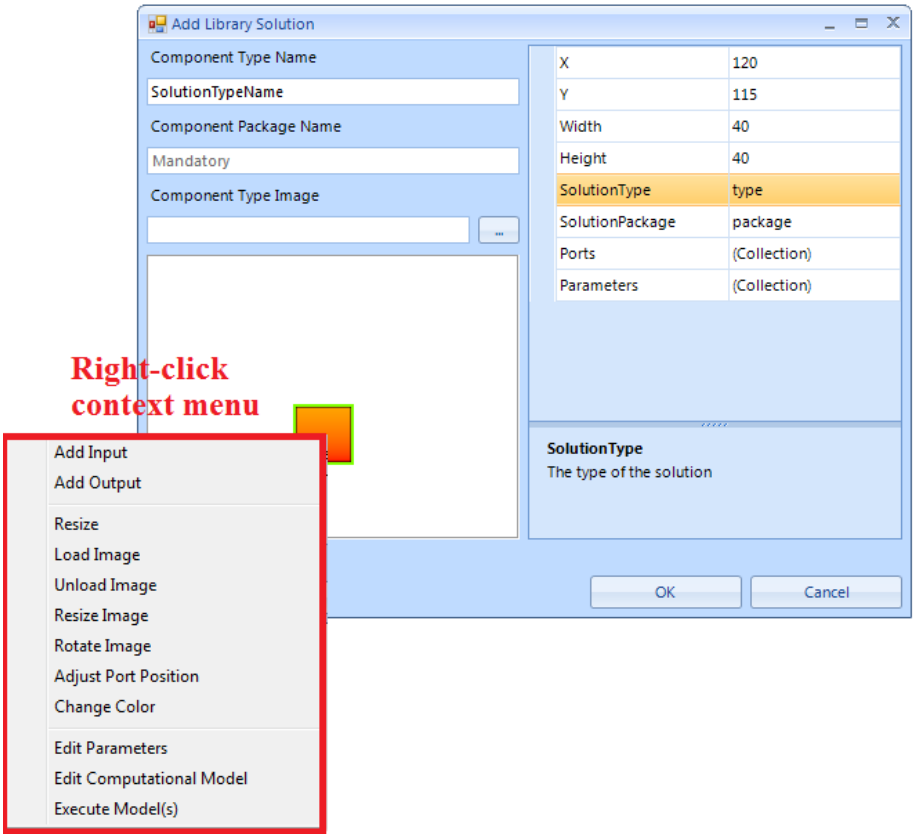


Figure C.10: *New solution type creation.*

C.1.4 Computational View

Once the architecture is defined, next, it is sized to find the system level performance. For this, a separate view, a computational view (CV) is proposed. It is linked to the architecture definition, and therefore, the architectural views mentioned above. The CV window is shown in Figure C.11. There is a toolbar at the top containing buttons 'Step-1', 'Step-2', 'Step-3', 'Layout', 'Execute', 'Group', and 'Aggregate'.

Three buttons ('Step-1', 'Step-2', 'Step-3') incorporate the methods developed for assessment of the architecture i.e. sequencing of sub-systems, construction of workflows,

and construction of a complete workflow, respectively. On clicking the 'Execute' button, the complete workflow produced in the 'Step-3' is executed. The 'Layout' button arranges the workflow graph. The 'Group' button clusters the models of the workflow in different collections corresponding to the components (of the sub-system). The 'Aggregate' button provides the interface where the user can specify an aggregation of non-additive variables between the levels of the system hierarchy.

The sub-systems sequence produced in the first step is displayed (in different stages) in the visualisation window on the right. For instance, for the example architecture, the sub-systems ECS and HPS are put in the second stage. Just below this window, there is a result window which displays the values of the parameters, after execution of the complete workflow. On the left of these two windows are the containers ('Sub-systems Level' and 'Complete workflow') which store the individual sub-systems and system workflows produced in 'Step-1', and the complete workflow obtained in 'Step-3'.

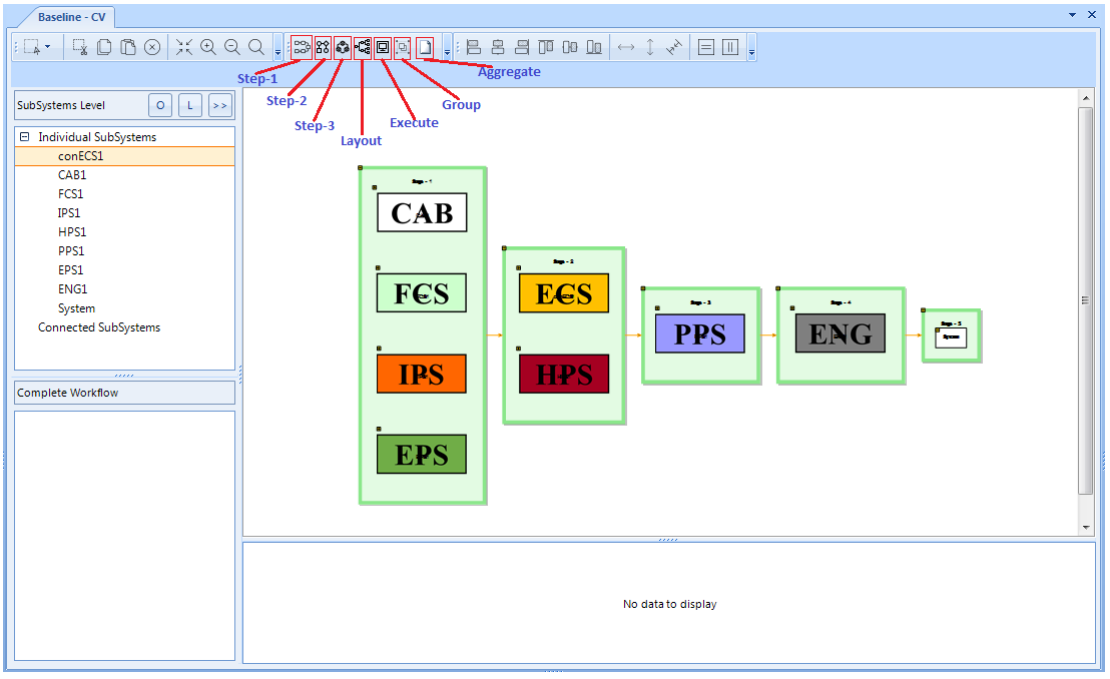


Figure C.11: *Computational view (CV).*

On clicking the button, 'Step-2', with the sub-system on the left panel 'Sub-system Level' being selected' of the CV, a new window pops up where the user can specify (independent) known and unknown parameters of the sub-system to construct the workflow. The window is shown in Figure C.12. Here, the user is provided with the

lists boxes (containing sub-system parameters), 'Unknown Parameters', 'Known Parameters' and 'Target Parameters'. The user has to specify, in the first activity, the known/unknown design information/parameters. In the second activity, all the possible matchings are produced by clicking the button 'Find all maximum matchings', and the list is displayed in the list box below it. This list contains the matchings with the least reversed variables at the top. The user can select the matching of the list to create 'Workflow' objects in the third activity by clicking the button, 'Create Workflow (for the selected matchings)'. All the workflows created are listed in the list box, 'List of workflow objects'. Furthermore, the created workflow contains any guess variables are displayed in 'Guess variables' list box. In addition, three are buttons ('Plot Workflow', 'Plot SCC', 'Execute') to plot and execute the workflows for testing if they execute. The results of the execution of the workflow are shown on the right bottom window.

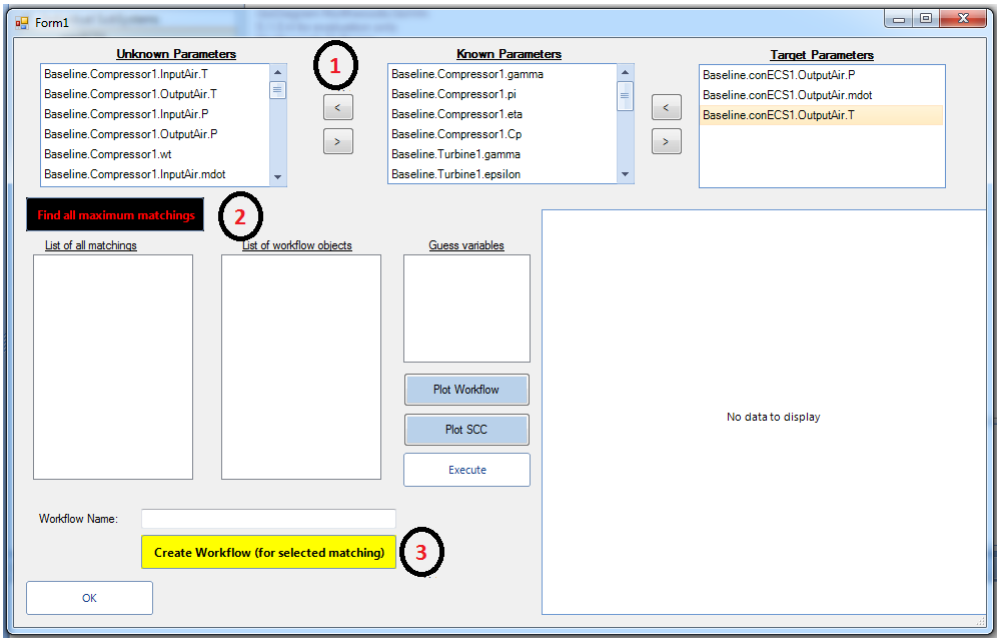


Figure C.12: *Step-2: construction of a workflow.*

The 'OK' button on this window adds the selected workflow object (from 'List of workflow objects' under the corresponding sub-system in the CV.

Similarly, workflows are created for all the sub-systems listed in the sub-systems container of the CV window. Once all the workflows are created, next, the complete workflow object is created by clicking the 'Step-3' button of the CV. The created workflow is added to the list box, 'Complete Workflow', of the CV. Being selected the complete

workflow in this list, on pressing the 'Execute' button, the selected workflow is executed and the results of the execution are displayed on the results window at the bottom of the CV. The results in the window can be gathered with the parameters of the component or parameters of the same type of port or dimension and so forth in a group, for easy access and better visualisation. The status of the execution is shown on 'Output' window of the main interface of AirCADia Architect (see Figure C.1).

In addition to the views (of the architecture) discussed above, a view which stores the functional reasoning knowledge in both functional and logical domain is proposed and referred to as 'Functional Logical View' (FLV). An example of the view is shown in Figure C.13. It captures the reasoning knowledge i.e. mapping and derived relations between functions and solutions in the form of a bipartite graph. These graphs along with Depth First Search (DFS) algorithm are employed to enable the use to trace the dependency between the elements of functional and logical views of the architecture.

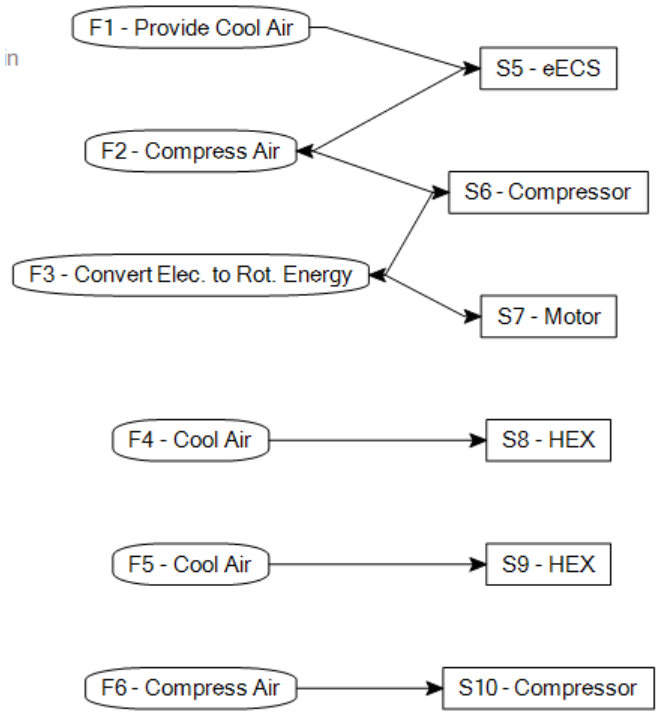


Figure C.13: *An example of FLV of the architecture.*

In addition to FLV of the architecture, the reasoning knowledge of different architectures is stored in a common database. This knowledge is also displayed in the form of a graph as shown Figure C.14. The user is facilitated an interactive mechanism to use this knowledge in a new architecture.

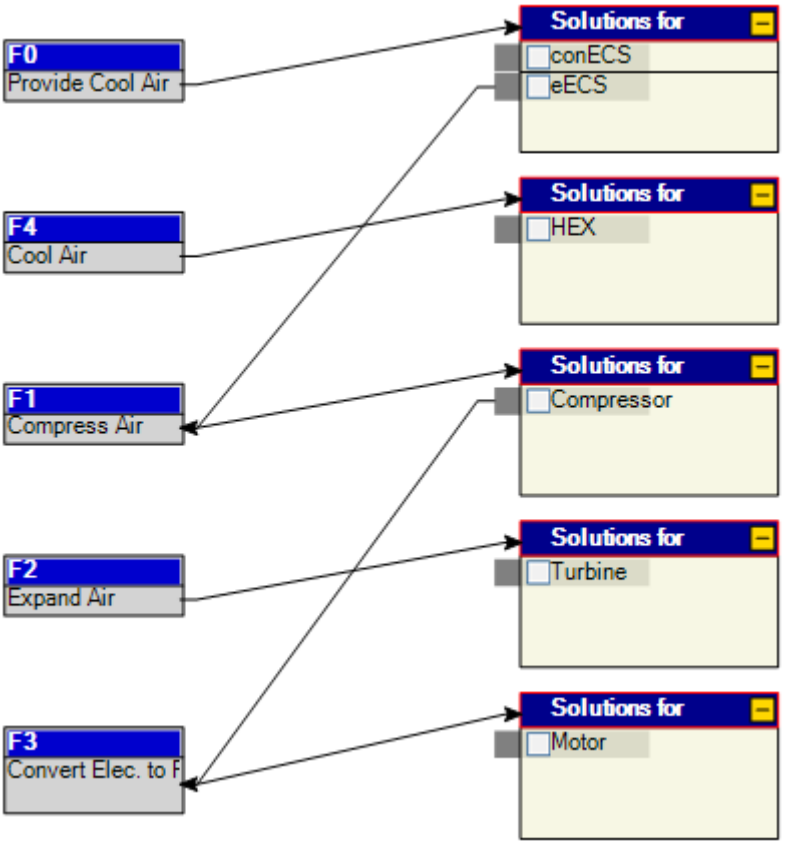


Figure C.14: *An example of functional reasoning knowledge database.*

All the relations that are specified by the user are also stored in the form of a graph, and indirect relations are calculated behind the scene and used to trace the inter-domain dependency among the elements.

C.2 Environmental Control System Models

In both, conventional and electrical ECS, Air Cycle Machine (ACM) remains the same. The schematic of Bootstrap Air Condition Pack is shown in Figure C.15. In case of conventional ECS, the air at location '1' comes from engine, whereas in electrical ECS, the air comes from the compressor driven by the electric motor.

The steady state models used to size the components of the pack are listed here.

Compressor (C):

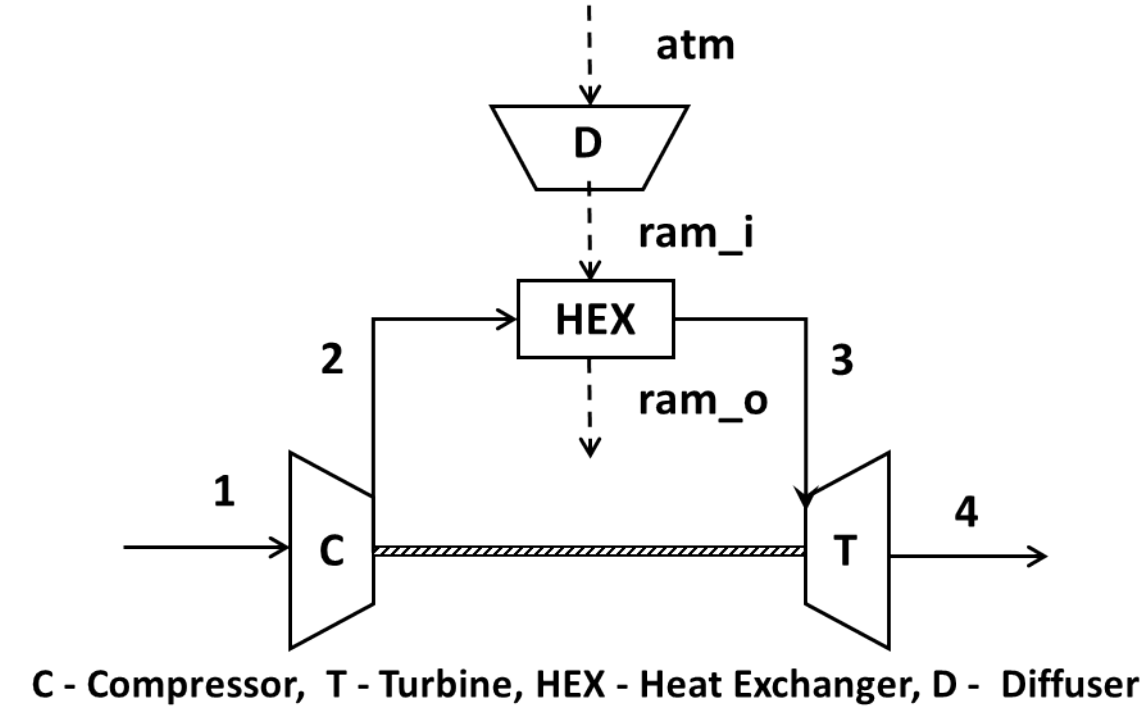


Figure C.15: *ACM Schematic.*

Considering adiabatic compression and conservation of mass across the compressor, the equations derived for temperature, pressure and mass flow rate at the output of the compressor are as follow.

$$T_2 = T_1(1 + \frac{1}{\eta_c} \times (CR^{\frac{\gamma-1}{\gamma}} - 1))$$

$$P_2 = P_1 \times CR$$

$$\dot{m}_1 = \dot{m}_2$$

Heat exchanger (HEX):

For the HEX, simplified equations are employed. Here, the pressure at the output of the heat exchanger is calculated by considering constant pressure drop across the HEX. The temperature at the output of the heat exchanger is calculated considering the efficiency of the HEX.

$$P_3 = P_2 - \Delta P_{HEX}$$

$$T_3 = T_2 - \eta_{HEX} \times (T_2 - T_{ram\_i}))$$

$$\dot{m}_2 = \dot{m}_3$$

Ram air stream: Due to ramming effect of the air, the temperature and pressure of the air, taken in the aircraft, is different from the temperature and pressure of the outside air. The temperature and pressure, therefore, depend on speed of the aircraft. As shown in the schematic of the ACM, the temperature, T_{ram_i} and pressure, P_{ram_i} calculated at the output of the diffuser are given by following equations. This ram air is employed to cool the pack air.

$$T_{ram\_i} = T_{atm} \times (1 + \frac{\gamma - 1}{2} \times M^2)$$

$$P_{ram\_i} = P_{atm} \times (\eta_{rd} \times (\frac{T_{ram\_i}}{T_{atm}} - 1))^{\frac{\gamma}{\gamma - 1}}$$

$$P_{ram\_o} = P_{ram\_i} \times (1 - \mu_c)$$

$$\epsilon = \frac{T_2 - T_3}{T_2 - T_{ram\_i}}$$

Turbine (T):

Considering adiabatic expansion and conservation of mass across the turbine, the equations of temperature, pressure and mass flow rate at the output of the compressor are as below.

$$\frac{1}{ER}^{\frac{\gamma - 1}{\gamma}} = 1 - \frac{1}{\eta_t} \times (1 - \frac{T_4}{T_3}) \Rightarrow T_4 = T_3 \times (1 - \eta_t \times (1 - ER^{-\frac{\gamma - 1}{\gamma}}))$$

$$P_4 = \frac{P_3}{ER}$$

$$\dot{m}_3 = \dot{m}_4$$

Compressor-Turbine coupling:

The compressor is driven using the work produced by the turbine. Therefore, both, work required and produced by the compressor and turbine respectively, should be equal. In achieving this, iterations are involved in the computations. To decide the compression

ratio of the compressor, firstly, compressor ratio is assumed, and all the above equations starting from the compressor are evaluated. At this point, if the turbine work is not equal to work required by the compressor, then again the compression ratio is changed, and this process is repeated.

$$\dot{W}_t = \dot{W}_c \Rightarrow \dot{m}_t \times C_p \times (T_3 - T_4) = \dot{m}_c \times (T_2 - T_1)$$

TABLE C.1: ACM parameters

Parameter	**Symbol**	**Value**
Compressor efficiency	η_c	0.75
Turbine efficiency	η_t	0.8
HEX max effectiveness	ϵ_{max}	0.8
Pressure drop across HEX	ΔP_{HEX}	10000

C.3 Application of Proposed Methods

C.3.1 Hypothetical Design Scenario

Screen captures of the application of the proposed methods (using prototype tool) on a test-case as per the steps (1 to 4) of the hypothetical design scenario shown in Figure C.16 are shown below.

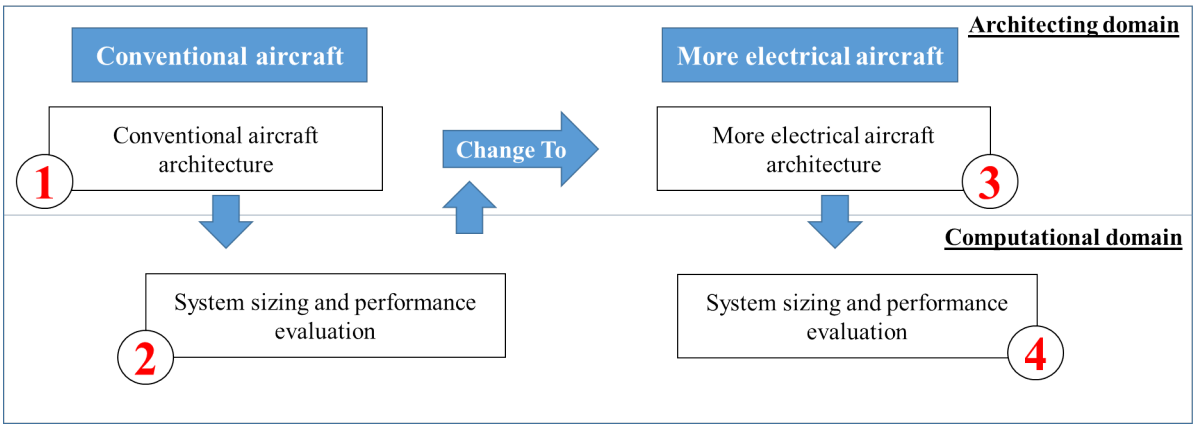


Figure C.16: *Hypothetical design scenario.*

C.3.1.1 Step 1:

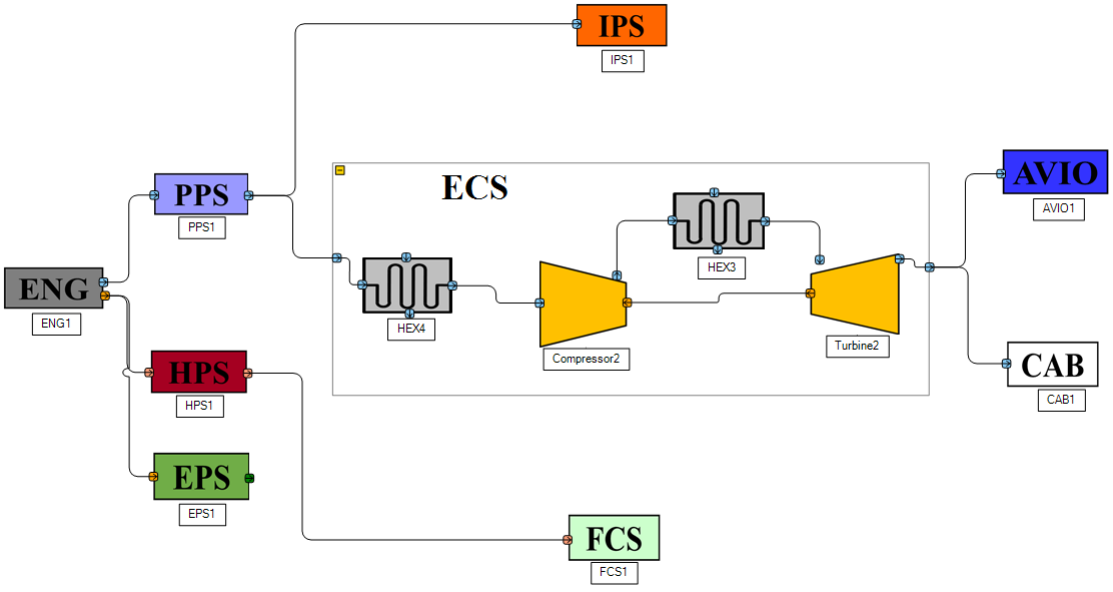


Figure C.17: *Conventional aircraft logical view.*

C.3.1.2 Step 2:

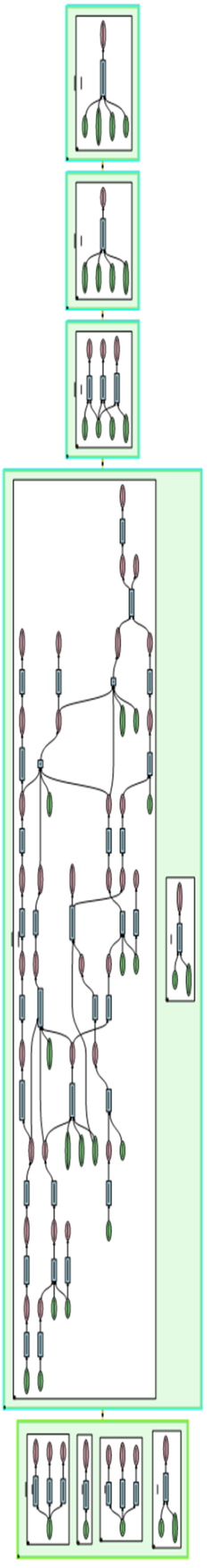


Figure C.18: *Computational view of conventional aircraft assessment.*

C.3.1.3 Step 3:

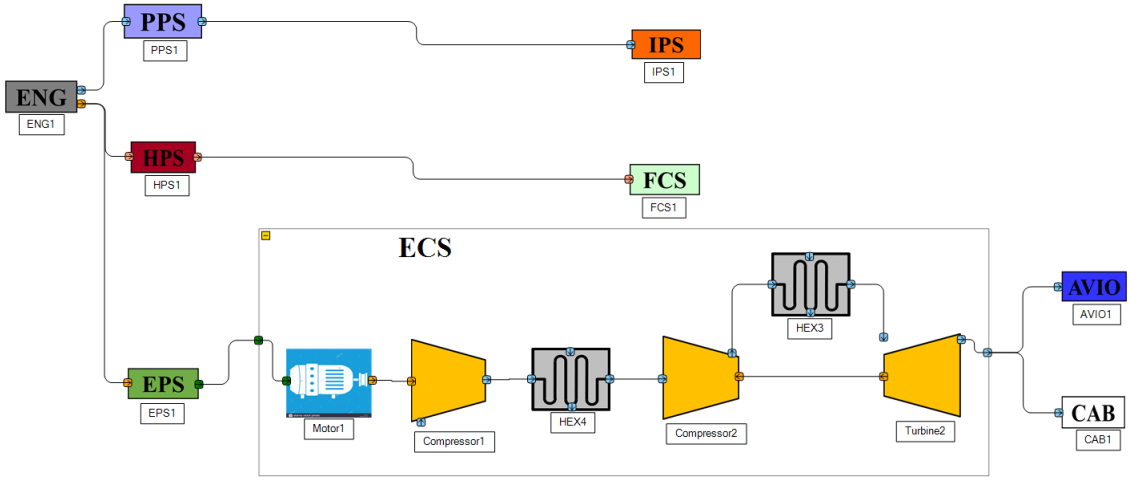


Figure C.19: *More electrical aircraft logical view.*

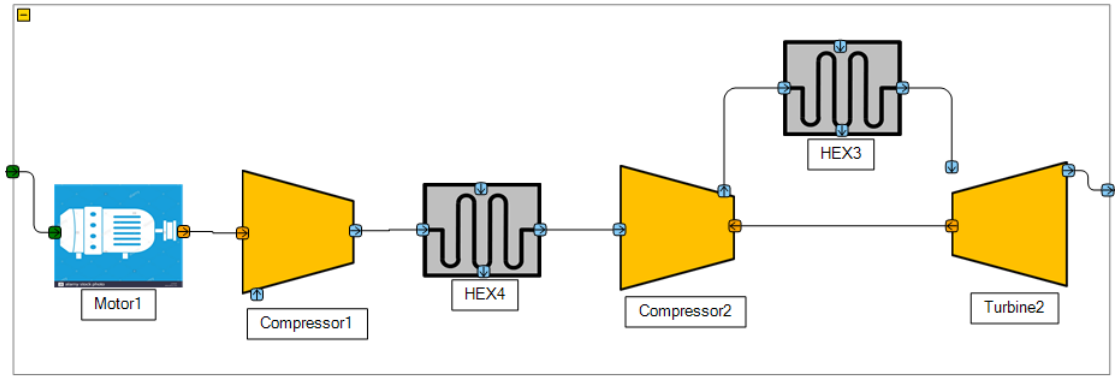


Figure C.20: *Electrical ECS architecture logical view.*

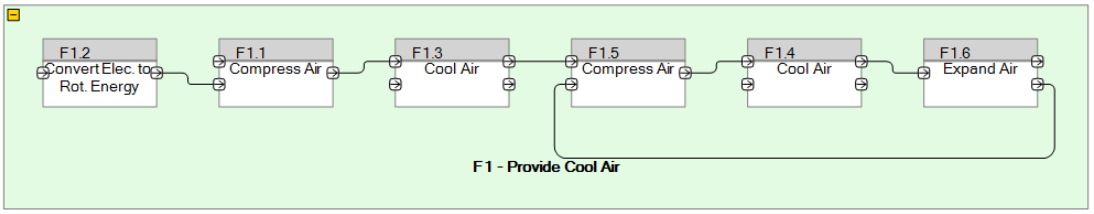


Figure C.21: *Electrical ECS architecture functional view.*

C.3.1.4 Step 4:

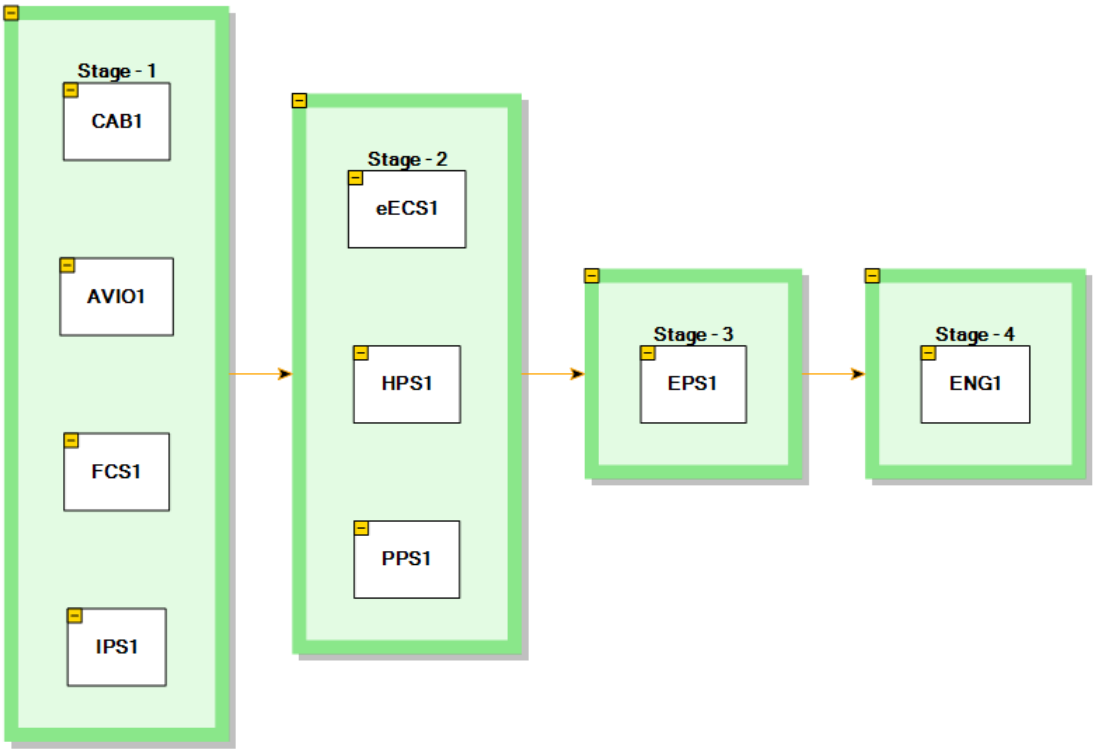


Figure C.22: *Sub-systems sizing sequence.*

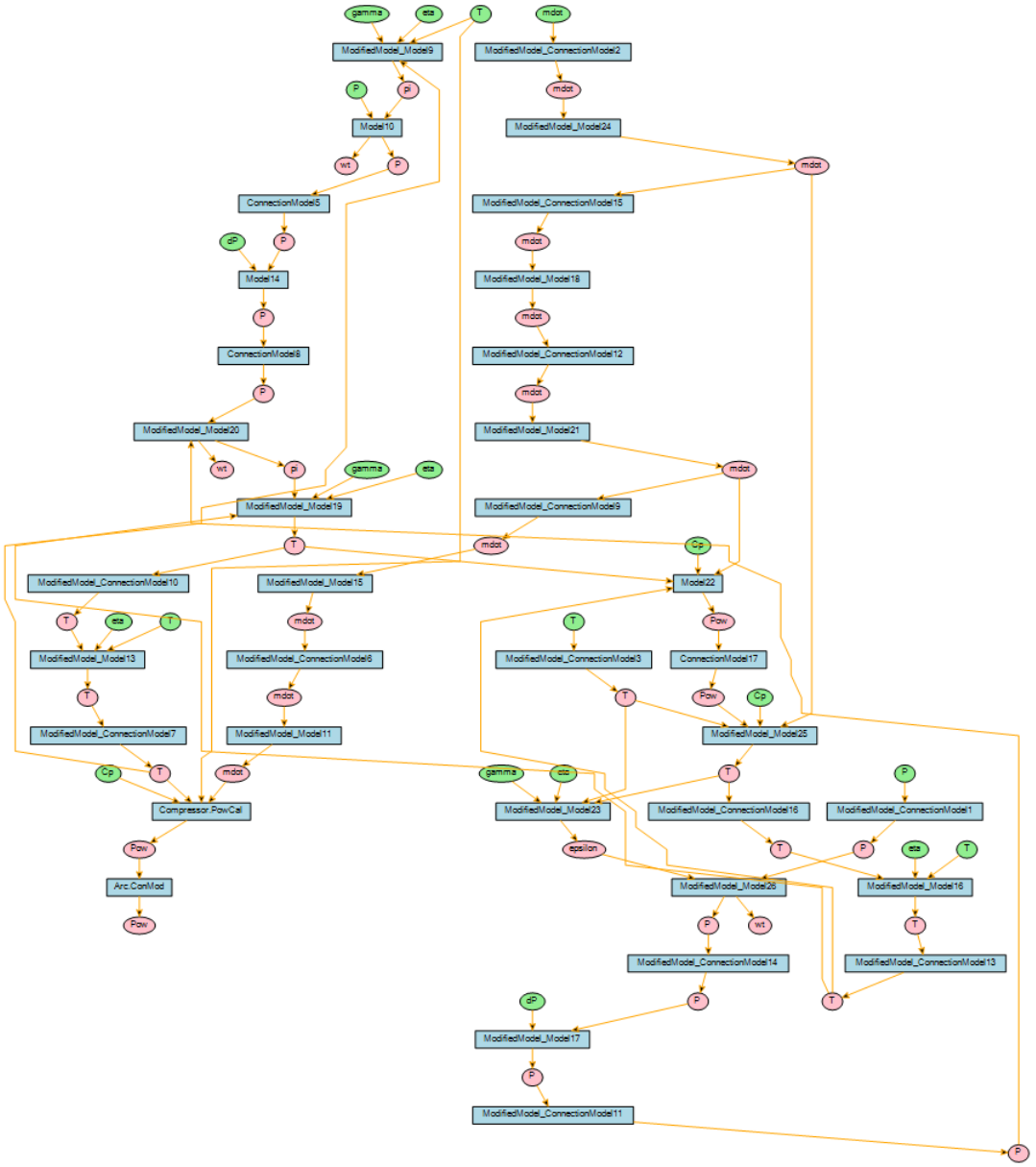


Figure C.23: *ECS sizing workflow.*

APPENDIX D

Illustration: DSM Sequencing

Below is the source-sink *DSM* obtained for the logical view shown in Figure 5.6. Here, rows and columns - 1, 2, 3, 4 and 5 represent sub-systems - SS-1, SS-2, SS-3, SS-4, and SS-5, respectively.

$$DSM_{5\times5} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Next step is to find $SCCs$ in the DSM which involves finding accessibility matrix - P as below.

$$P_{5\times5} = \sum_{i=1}^{5} DSM^{i}_{5\times5} = \begin{bmatrix} 31 & 31 & 26 & 26 & 16 \\ 31 & 31 & 31 & 31 & 26 \\ 0 & 0 & 5 & 0 & 15 \\ 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 5 \end{bmatrix}$$

Then, non-zero elements of the accessibility matrix - P is replaced with 1 to obtain binary accessibility matrix and transpose (P^T) of this matrix as below.

$$P_{5\times5} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, P^{T}_{5\times5} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Next step is to calculate $P \circ P^T$,

$$P \circ P^T = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Here, first two rows of $P \circ P^T$ are similar, therefore there is a SCC containing sub-systems associated with these rows; these sub-systems are $SS-1$ and $SS-2$. Now, the sub-systems associated with this SCC are collapsed and new DSM showing the two sub-system as single representative element in the DSM. The collapsed DSM - $DSM^{c}_{4\times4}$ is shown below.

$$DSM^{c}_{4\times4} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Now, on the collapsed DSM, level-division theorem is applied. The step by step application of the theorem on above collapsed DSM is given below.

Level-1: $L = 1, E_{0_{4\times1}} = (1,1,1,1)^T$

$$DSM^c_{4\times1} = DSM^c_{4\times4} \cdot E_{0_{4\times1}} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \\ 1 \\ 1 \end{bmatrix}$$

$\therefore$ **Level-1** = { **SS-4, SS-5** }

Level-2: $L = 2, E_{2_{4\times1}} = (1,1,0,0)^T$

$$DSM^c_{4\times1} = DSM^c_{4\times4} \cdot E_{1_{4\times1}} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$\therefore$ **Level-2** = { **SS-3** }

Level-3: $L = 3, E_{3_{4\times1}} = (1,0,0,0)^T$

$$DSM^c_{4\times1} = DSM^c_{4\times4} \cdot E_{1_{4\times1}} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$\therefore$ **Level-3** = { **SCC** } = { **SS-1, SS-2** }

Level-4: $L = 4, E_{4_{4\times1}} = (0,0,0,0)^T$, Stop the process as $E_{4_{4\times1}}$ is zero vector.

APPENDIX E

Algorithms

E.1 Maximum Matching Enumeration Algorithm

E.1.1 Definition and Terminology

Bipartite graph: It is a special kind of a graph whose vertices are decomposed into two distinct vertex sets, and no two vertices from the same set are adjacent. The bipartite graph (BG) is represented as below.

$$BG = G(V1, V2, E) \tag{E.1}$$

Here,

$$M \cap V = \emptyset \tag{E.2}$$

where, $\emptyset$ is a null set.

In addition, for an edge, e, described by its end nodes, a and b, if $e = (a,b) \wedge e \in E$ then,

$$(a \in V1 \wedge b \in V2) \oplus (b \in V1 \wedge a \in V2) \tag{E.3}$$

Here, $\oplus$ - is an 'exclusive or' operator.

Matching: It is a set of graph-edges such that no two edges share a vertex. That is, each vertex of the matching is associated with only one edge of the matching.

In case of above bipartite graph, a matching is a subgraph of BG, M, where every node has a degree of 1. In other words, the matching is a set of distinct edges sharing no endpoints.

Maximum matching: For a given graph, it is a matching that contains the largest possible number of edges in it. It is the matching with maximum cardinality. It is possible to have more than one matching with the same cardinality.

Perfect matching: It is a matching where every vertex is matched. All perfect matchings are maximum matchings.

E.1.2 Maximum Matching: Hopcroft-Karp Algorithm

E.1.3 Maximum Matching Enumeration Algorithm

The algorithms employed for enumerating maximum matchings (proposed by Uno33) of a bipartite graph are shown below.

1) *ENUM_MAXIMUM_MATCHINGS* [94]

ALGORITHM ENUM_PERFECT_MATCHINGS (G)
Step 1: Find a perfect matching M of G and output M. If M is not found, stop.
Step 2: Trim unnecessary edges from G by a strongly connected component decomposition algorithm with $D(G, M)$
Step 3: Call ENUM_PERFECT_MATCHINGS_ITER (G, M).

2) *ENUM_MAXIMUM_MATCHINGS_ITER* [94]

ALGORITHM ENUM_PERFECT_MATCHINGS_ITER (G, M)
Step 1: If G has no edge, stop.
Step 2: Choose an edge e.
Step 3: Find a cycle containing e by a depth-first search algorithm.
Step 4: Find a perfect matching M' by exchanging edges along the cycle.Output M'
Step 5: Trim unnecessary edges from $G^+(e)$.
Step 6: Enumerate all perfect matchings including e by ENUM_PERFECT_MATCHINGS_ITER with the obtained graph and M.
Step 7: Trim unnecessary edges from $G^-(e)$.
Step 8: Enumerate all perfect matchings not including e by ENUM_PERFECT_MATCHINGS_ITER with the obtained graph and M'.

E.2 Application of Sugiyama Algorithm for Layered Graph Drawing to Sequencing SCC Models

For demonstration purpose, ECS workflow containing SCC is considered. The SCC of the workflow contains seven models as shown in Table E.1. With this sequence of models, there are four feedback variables which can be obtained from the number of occurrences of '1's below main diagonal of the model to model DSM of the SCC models.

In order to reduce these feedback variables in the SCC, the models of the SCC are required to be re-organised. For this purpose, the workflow (directed graph) of ECS is layout (or plotted) using Sugiyama algorithm[98] as shown in Figure E.1. The SCC part of the workflow is drawn by the red line.

TABLE E.1: Models contained in the SCC.

SCC Models
Model39
Modified Model42
Modified Model44
Modified Model50
Modified Connection Model14
Modified Connection Model 17
Connection Model18

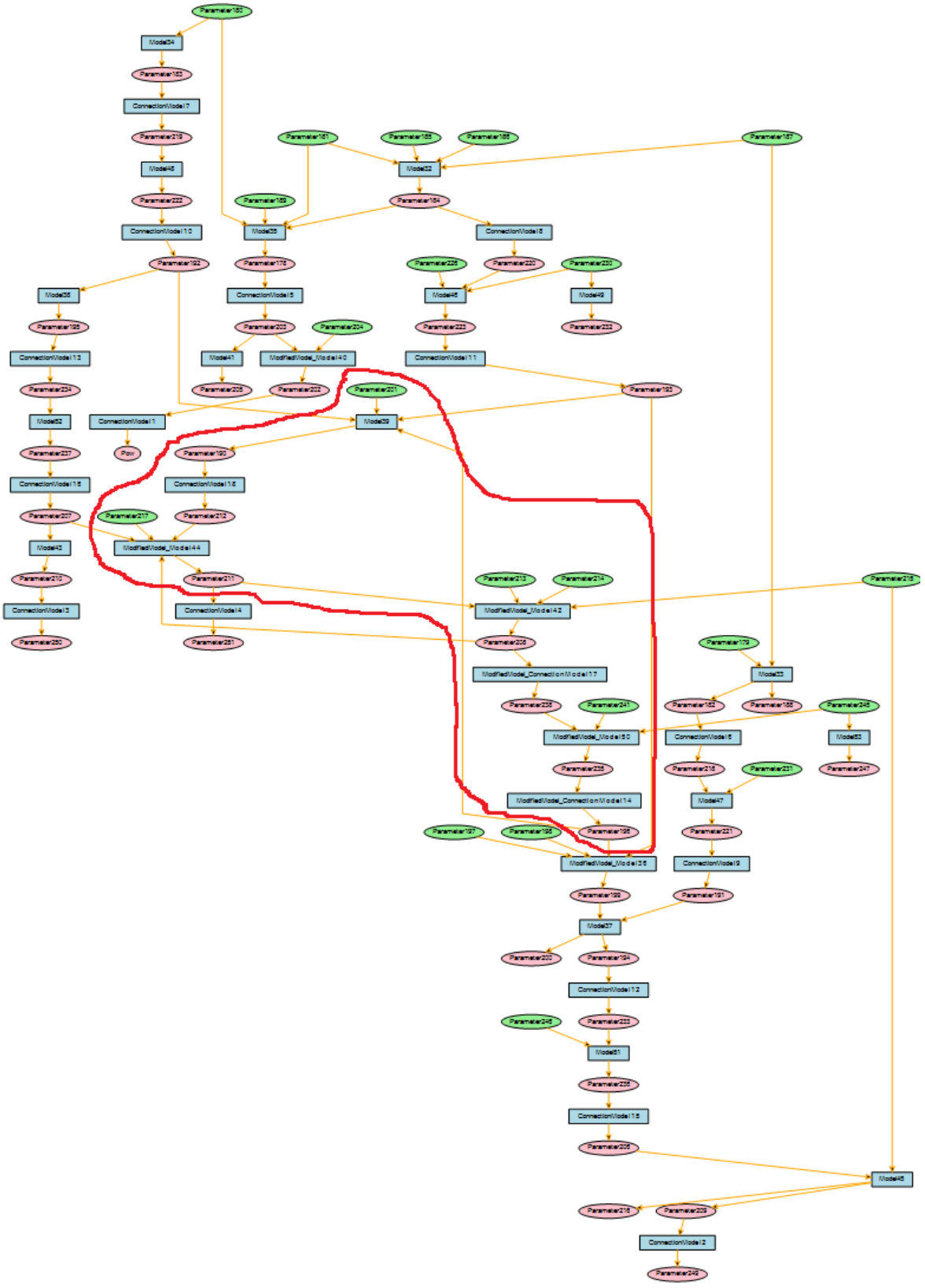


Figure E.1: *ECS workflow.*

Instead of performing layout on the complete ECS workflow, only part of the workflow, i.e., SCC is plotted using Sugiyama algorithm, to reduce computational cost. The plotted SCC is shown in Figure E.2.

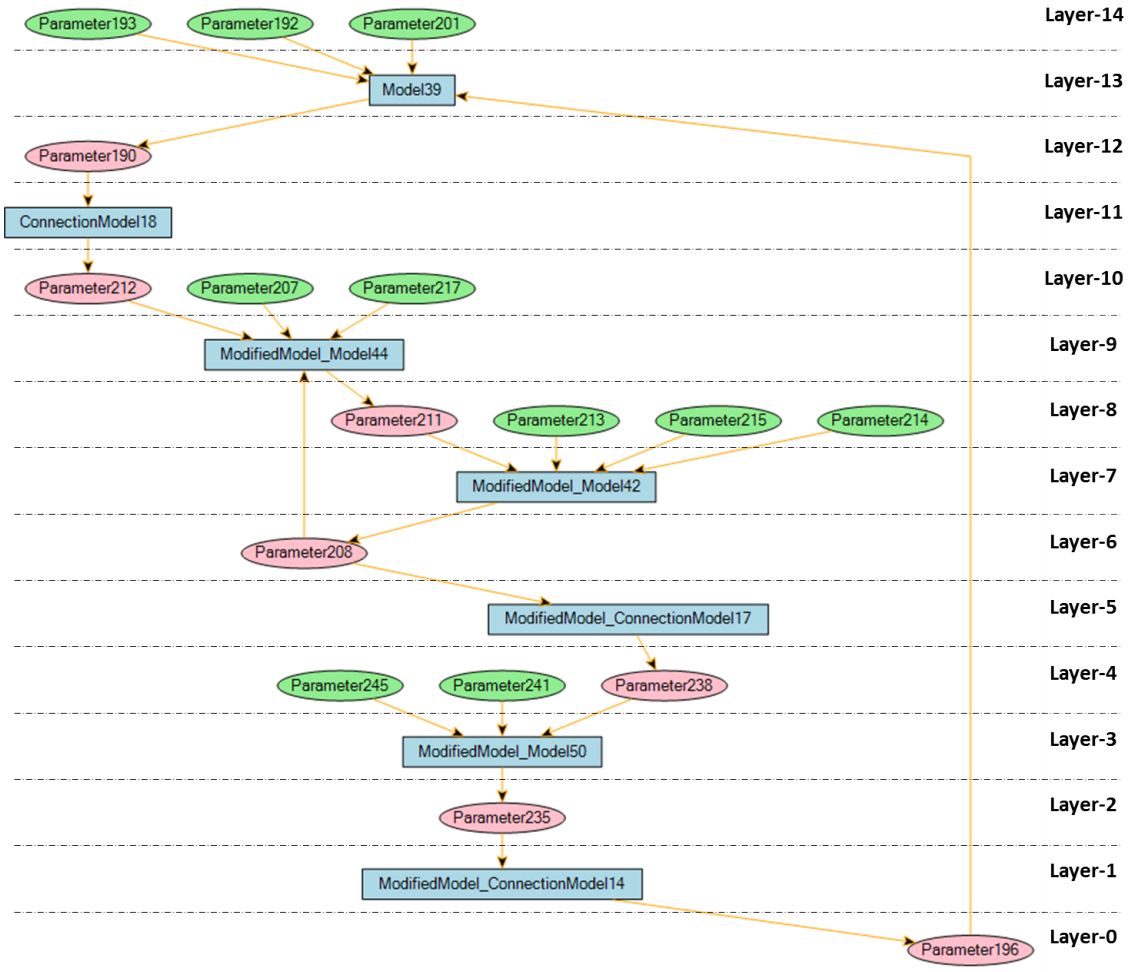


Figure E.2: *Layered layout of SCC of ECS workflow.*

The layer information of the workflow layout is employed to obtain the sequence. The layers of the SCC models and variables are shown in Table E.2.

For sequencing of the models, only information related to models is extracted as shown in Table E.3.

TABLE E.2: Layer information of the SCC models' layout.

Variable/Model	Layer	Column
Parameter193	14	4
Parameter192	14	11
Parameter201	14	18
Parameter213	8	23
Parameter215	8	30
Parameter214	8	37
Parameter207	10	11
Parameter217	10	18
Parameter245	4	14
Parameter241	4	21
Parameter196	0	41
Parameter190	12	4
Parameter208	6	12
Parameter211	8	16
Parameter212	10	4
Parameter235	2	21
Parameter238	4	28
Model39	13	18
Modified Model44	7	23
Modified Model44	9	12
Modified Model50	3	21
Modified Connection Model14	1	21
Modified Connection Model17	5	26
Connection Model 18	11	4

TABLE E.3: Models of SCC.

Model	Layer	Column
Model39	13	18
Modified Model42	7	23
Modified Model44	9	12
Modified Model50	3	21
Modified Connection Model14	1	21
Modified Connection Model17	5	26
Connection Model18	11	4

The layers of Table E.3 are arranged in descending order and corresponding models'

sequence is obtained. The sequence of the models is shown in Table E.4. With this sequence, the number of feedback variables is two which is lower than the number before sequencing of SCC models.

TABLE E.4: Sequence of models in the SCC.

Model	Layer	Column
Model39	13	18
Connection Model18	11	4
Modified Model44	9	12
Modified Model42	7	23
Modified Connection Model17	5	26
Modified Model50	3	21
Modified Connection Model14	1	21

APPENDIX F

Evaluation Session

F.1 Debriefing

Thank you very much for your time today to assist with this research. As mentioned earlier the data provided by you will be treated confidentially and you are free to withdraw or change your responses for a period of up to 7 days from today. If you need to change anything you can let me know by contacting me through email or phone (details are on page 1 of this form).

If you are happy for me to contact you by email or phone at a later date if I have missed out important points or need to clarify any of your responses, please complete your contact details below.

Thank you

Participant's contact details: Tel: Email:	

F.2 Informed Consent Form

Title of the Project:	Component-driven Computational Design of Complex Engineering Systems
Name of the Researcher:	Yogesh Hanumant Bile
Researcher's Contact Details:	07810671823
Participant No:	
Date:	

1. I confirm that I have been informed about the aim and objectives of this research project and agree to give my inputs.

2. I understand that all personal information that I provide will be treated with the strictest confidence and my name will not be used in any report, publication or presentation. I have been provided with a participant number to ensure that all raw data remains anonymous.

3. I understand that the information I provide will be used by Cranfield University for the purpose of research only. The data will be stored on a secure network accessed only by authorised users in accordance with the Data Protection Act (1998).

4. I understand that the results of the research may be published in scientific journals, and an anonymised version of the data may be published in support of these results.

5. I understand that I am free to withdraw from this project at any stage during the session simply by informing a member of the research team, for whom contact details have been provided. I also understand that I can also withdraw my data for a period of up to 7 days from today, as after this time it will not be possible to identify my individual data from the aggregated results.

I confirm I have read and fully understand the information provided on this form and therefore give my consent to taking part in this research.

Participant's signature		Date:	
Participant's name			
Researcher's signature		Date:	

F.3 Questionnaire

Questionnaire

Please answer the following questions keeping the context of the presented research in mind.

1. To what extent do you agree with the following statements (tick ✓ in the appropriate box):

Statement	Strongly disagree	Disagree	Neither agree or disagree	Agree	Strongly agree
a. The proposed method would enhance interactivity during the Architecture Definition (AD) and Architecture Assessment (AA) processes					
b. The proposed method is likely to enable me to reduce the number of time-consuming manual activities in AD & AA.					
c. It is reasonable to assume that component computational models would be available during conceptual design.					

2. Please rate the proposed method against the following metrics:

   a. Practicality ☐1 1=poor ☐2 2=fair ☐3 3=average ☐4 4=good ☐5 5=excellent

3. Open-ended comments (use the back of the sheet if you need more space):

   a. What specifically, do you think, the method does well (in the context of the research)?

b. How do you think the method could add value in the conceptual design stage?

c. How do you think the method could be improved?

Your Name: _____ (Optional)
Your Department: _____
Years of Experience: _____

Thank you very much for your time! It is greatly appreciated.

F.4 Feedback of the Experts on Open-ended Questions