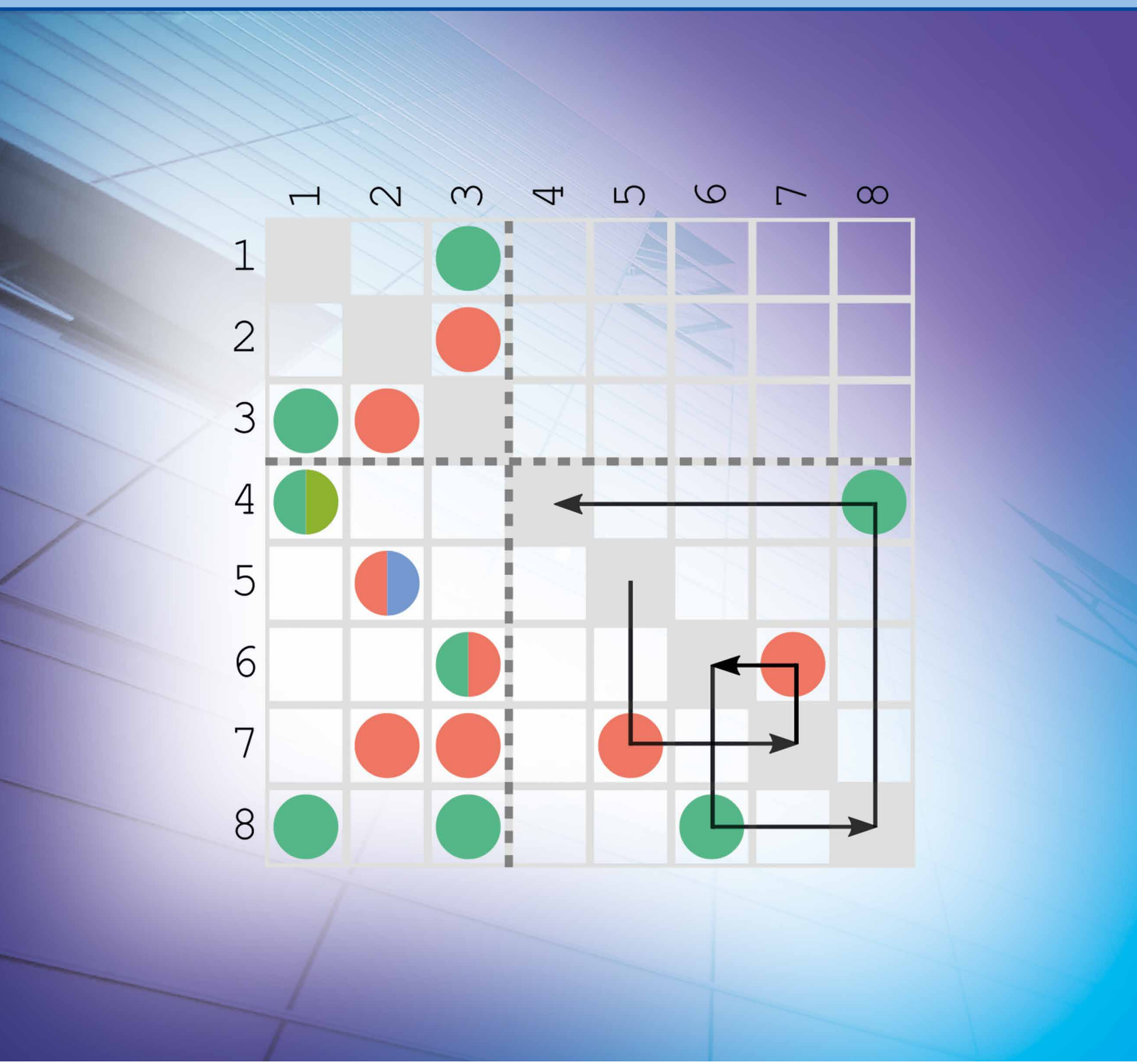
DSPE MIKRONIEK
2023 (VOL. 63) ISSUE 3
PROFESSIONAL JOURNAL ON PRECISION ENGINEERING



- THEME: MODEL-BASED SYSTEMS ENGINEERING
- BRIDGING THE GAP BETWEEN REQUIREMENTS ENGINEERING AND SYSTEMS ARCHITECTING
- CALCULATING THE TOTAL STIFFNESS OF COMBINED SPRINGS
- DSPE CONFERENCE ON PRECISION MECHATRONICS 2023 PREVIEW

MIKRONIEK IS A PUBLICATION OF THE DUTCH SOCIETY FOR PRECISION ENGINEERING – WWW.DSPE.NL

PUBLICATION INFORMATION

Objective
Professional journal on precision engineering and the official organ of DSPE, the Dutch Society for Precision Engineering. Mikroniek provides current information about scientific, technical and business developments in the fields of precision engineering, mechatronics and optics. The journal is read by researchers and professionals in charge of the development and realisation of advanced precision machinery.

DSPE
YOUR PRECISION PORTAL

Publisher
DSPE
Julie van Stiphout
High Tech Campus 1, 5656 AE Eindhoven
PO Box 80036, 5600 JW Eindhoven
info@dspe.nl, www.dspe.nl

Editorial board
Prof.dr.ir. Just Herder (chairman, Delft University of Technology),
Servaas Bank (VDL ETG), B.Sc.,
Maarten Dekker, M.Sc. (Philips),
Otte Haitsma, M.Sc. (Demcon),
dr.ir. Jan de Jong (University of Twente),
Erik Manders, M.Sc. (Philips Engineering Solutions),
dr.ir. Pieter Nuij (MaDyCon),
dr.ir. Ioannis Proimadis (VDL ETG),
Maurice Teuwen, M.Sc. (Janssen Precision Engineering)

Editor
Hans van Eerden, hans.vaneerden@dspe.nl

Advertising canvasser
Gerrit Kulsdom, Sales & Services
+31 (0)229 – 211 211, gerrit@salesandservices.nl

Design and realisation
Drukkerij Snep, Eindhoven
+31 (0)40 – 251 99 29, info@snep.nl

Subscription
Mikroniek is for DSPE members only.
DSPE membership is open to institutes, companies, self-employed professionals and private persons, and starts at € 80.00 (excl. VAT) per year.

Mikroniek appears six times a year.

© Nothing from this publication may be reproduced or copied without the express permission of the publisher.

ISSN 0026-3699

The cover image (the second decomposition level of a component-function multi-domain-matrix) is courtesy of Ratio. Read the article on page 5 ff.

IN THIS ISSUE

THEME: MODEL-BASED SYSTEMS ENGINEERING

05
Bridging the gap between requirements engineering and systems architecting
To support the transition from document-driven to model-based systems engineering, the open-source Elephant Specification Language (ESL) has been developed, for writing highly structured system specifications from which system architecture models are automatically derived.

10
Mathematical approach of high-tech systems design
Axiomatic Design enables the creation of good system designs by improving the causality of system functionality, their physical realisation, and manufacturing processes.

16
System behaviour prediction by modelling and simulation
Rationale for model-based systems engineering.

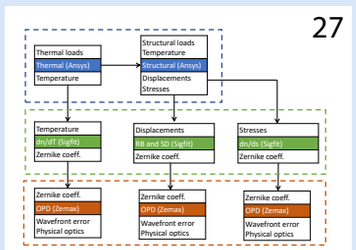
24
Comprehensive Systems Engineering Education
Laying the foundation for the Dutch approach.

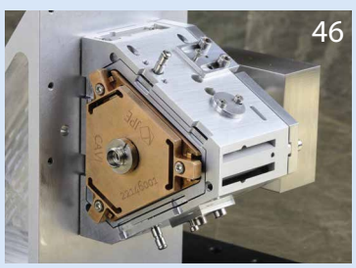
27
'STOP' to optimise performance
Predicting optical system performance requires the integration of separate analyses, known as Structural, Thermal, Optical Performance – STOP.

32
Design Principles – Calculating the total stiffness of combined springs
Multi-DoF systems with multiple stiffnesses can show both series and parallel behaviour.

38
Event preview – DSPE Conference on Precision Mechatronics 2023
The theme of this fifth edition is 'Rethinking the system' and registration is still open.

46
Design & Realisation – Cryogenic active vibration isolator
A compact and lightweight one-DoF module which is meant to be integrated into a multi-DoF isolation solution.





FEATURES

04 EDITORIAL
Gerrit Muller, professor of Systems Engineering in Norway and senior research fellow at TNO-ESI, on complexity and interoperability challenges in systems engineering.

45 TAPPING INTO A NEW DSPE MEMBER'S EXPERTISE
ACS Motion Control – high-performance OEM motion-control system solutions

51 DSPE
Including: Volunteer in the spotlight, Marc Vermeulen.

53 UPCOMING EVENTS
Including: DSPE Knowledge Day Cryogenics.

54 ECP2 COURSE CALENDAR
Overview of European Certified Precision Engineering courses.

55 NEWS
Including: Piet van Rens, doyen of Dutch design principles, retires.

WHY IS EVERYONE TALKING ABOUT MBSE ANYWAY?

All my different sub-persons come across the term MBSE (model-based systems engineering): associate editor of Systems Engineering, fellow of INCOSE, professor in Norway, and researcher at ESI (TNO). These encounters have several different patterns:

- A large group of MBSE "advocates" look particularly concerned if a diagram does not use SysML.
- A large group of executives and developers hear the term and are afraid of missing out on something important.
- A group of older (?) professionals are disturbed by SysML and the high level of hype.

So, it is high time to bring some light into the darkness. Let's state some facts. The first simple fact is that for decades we have benefited from a wide range of digital tools for almost all our engineering work, such as requirements, configuration, and version management, software development tools, CAD-E, CAD-M, PLM, ERP, and the alphabet soup can go on for a long time. All those tools need a formalism and a representation to work. The result of engineering is that we create the technical production documentation, which is practically completely digital and largely formalised. Can anyone imagine developing a top-of-the line Intel CPU without extensive digital support?

Today's pain in developing complex systems is in various dimensions. A rather mundane problem is that although most information is digital somewhere, many steps in the entire development and product lifecycle require "manual" interventions; interoperability between many of the tools is still problematic. Part of the interoperability challenge is that many digital artifacts (models?) are discipline-oriented. How will all these artifacts form an integral, fluent solution? There is a clear need for integrating technology (with its formalism and representations) that streamlines the entire lifecycle.

A second fact is that complexity of the problem space causes another pain. Many systems that we make have a wide variety of stakeholders with an even wider range of needs and interests. These needs and interests go beyond technology, e.g. political, economic, social, environmental, and legal considerations are at play. Domain knowledge (defence, transportation, healthcare, etc.) plays a big role. The heterogeneity of stakeholders and considerations, in combination with ambiguity and fluidity of human interaction, requires communication, cooperation, influencing, and many other "soft" skills. This part of the pain is ill-served with formalism; it requires a broad spectrum of representations used in social interaction.

Our challenge is that we have to resolve both pains, in a way that both worlds function together. The digital interoperability will need something MBSE-like. Critical thinkers wonder whether the current SysML is a proper fit; that discussion goes beyond an editorial. The problem space complexity and the stakeholder interaction require a completely different paradigm of communication and visualisation, which I call conceptual modelling. Lastly, we need a way of working that is connecting both paradigms, continuously, from conception until decommissioning. This way of working is systems engineering.

Gerrit Muller
Professor of Systems Engineering, University of South-Eastern Norway,
and senior research fellow, Embedded Systems Innovations by TNO
gerrit.muller@gmail.com, www.gaudisite.nl

BRIDGING THE GAP BETWEEN REQUIREMENTS ENGINEERING AND SYSTEMS ARCHITECTING

New methods and software tools have been designed to support the transition from document-driven to model-based systems engineering. Though bringing improvements, these methods and tools bring along several practical challenges. To resolve the issues, Ratio Computer Aided Systems Engineering continues the development of the open-source Elephant Specification Language (ESL), for which the foundations were laid at Eindhoven University of Technology. ESL is a language to write highly structured system specifications from which system architecture models are automatically derived. It has been designed from an engineering perspective rather than an information management perspective, with the ultimate goal of bridging the gap between requirements engineering and systems architecting.

TIM WILSCHUT

Introduction

In recent years, several methods and software tools have emerged to support the transition from document-driven systems engineering to model-based systems engineering. Though being an improvement from fully document-driven engineering, these methods and tools bring along several practical challenges.

Firstly, most of these methods and tools work like databases in which pieces of information, such as requirements, are manually labelled and linked to other pieces of information, such as elements of the product breakdown structure (PBS), to keep track of all the information (i.e. what relates to what) that is being produced during the course of a development project. This manual labelling and linking of requirements to breakdown structure elements results in a heavy administrative workload for systems engineers and architects.

With stringent deadlines it cannot be avoided that corners are cut here and there. Moreover, to create these links one requires a deep understanding and an overview of the entire system. As the complexity of systems has been increasing in recent decades, it is impossible for a person to have such a complete understanding and overview. Moreover, continuous development and many unknowns complicate matters even further.

Secondly, the quality and consistency in labelling and linking completely relies on 'good behaviour' of those who create the labels and links. There are no means to, for example, formally check whether a requirement really relates to the elements it has been linked to. Instead, one usually resorts to expert reviews, which quickly lead to lengthy discussions.

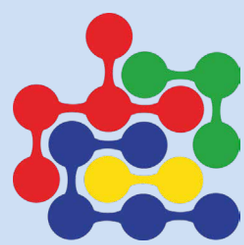
AUTHOR'S NOTE

Tim Wilschut is a co-founder of Ratio Computer Aided Systems Engineering, located in Eindhoven (NL), and hybrid lecturer at Eindhoven University of Technology (TU/e). He studied Mechanical Engineering at TU/e, where he also obtained his Ph.D.; his thesis was titled "System specification and design structuring methods for a lock product platform" [1]. Building on this work and the collaboration with the TU/e High-Tech Systems Center, Ratio develops systems engineering methods and tools. Wilschut does so together with Ratio's other co-founder, Tiemen Schuijbroek, since 2018, when they had finished their Ph.D. and M.Sc. thesis, respectively.

t.wilschut@ratio-case.nl
t.j.l.schuijbroek@ratio-case.nl

Ratio

Ratio Computer Aided Systems Engineering is a company that specialises in the development of methods and tools for requirements engineering and the modelling, analysis, and design of system architectures and product portfolios. Ratio has experience with modelling and analysing a wide variety of systems, ranging from locks and bridges to nuclear fusion power plants. Recently, Ratio set up a partnership with Quootz to allow customers to directly apply Ratio's product-portfolio analysis methods within the Quootz product configurator software.



WWW.RATIO-CASE.NL
WWW.QUOOTZ.NL

Thirdly, most tools focus on information management, not on the quality of the information being managed. As such, these tools simplify the management of large volumes of information. The information itself, however, may be just as vague and ambiguous as in a fully document-driven approach.
These combined challenges often result in a linking structure that is inconsistent and incomplete, does not provide a model of the systems architecture, and therefore has little value in everyday engineering practice.

Therefore, systems engineers and architects often resort to graphical modelling tools to create systems architecture models manually. In turn, these models must be kept up-to-date and consistent across the board as well as with their related requirement specifications, which increases the administrative workload even further and results in a gap between requirements engineering and systems architecting.

Six blind men and the elephant

When creating systems architecture models, systems engineers and architects will often find themselves in lengthy discussions on what the systems architecture really is. These discussions often resemble the parable of the six blind men that went to see an elephant. As the story goes, they explored the elephant by touch and disputed long and loud about what an elephant is, until the prince came out. "Be quiet now," he said, "because you are all in the right and you are all in the wrong." In real life, however, there is usually no prince to be found.

To resolve these issues, the open-source Elephant Specification Language (ESL) [1] [2] was created. ESL is a simple, highly structured formal language for defining a PBS and all functional, behavioural, and design requirements in a consistent and concise manner. As it is natural-language-based, it is readable by any engineer. Yet, it is sufficiently structured to allow the ESL compiler to automatically derive dependencies (links) between functional requirements, behavioural requirements, design requirements, and elements of the PBS based on mathematical rules. This network of dependencies defines the systems architecture.

Automated dependency derivation reduces the risk of human error drastically as dependencies cannot be forgotten. Moreover, it reduces the human workload tremendously. For example, at the Dutch Institute For Fundamental Energy Research (DIFFER) [3], ESL is being used to create the conceptual design of a pulsed-laser-deposition research cluster. At the time of writing, the research cluster specification defines a PBS comprising three layers containing 161 elements. These elements are subject to nearly a thousand requirements between which over 13,000 dependencies are automatically derived in less than a second, which allows for efficient systems-architecting modelling and analysis.

The text-based format allows one to easily manage ESL files using version control software such as Git and SVN, which has been used for change management in software development for decades.

A break with convention

ESL has been designed from an engineering perspective rather than an information management perspective. In other words, it has been designed to aid systems engineers and architects in designing the system by allowing one to automatically generate systems architecture models from the requirement specifications. Perhaps counterintuitively, it therefore deliberately deviates from several classical systems engineering concepts.

One decomposition to rule them all

Firstly, in classical systems engineering it is often advocated to use separate system, function, and requirement decomposition trees of which the elements are linked to another. In practice, however, systems engineers will often experience that it is very hard to create separate function and requirement trees and that when they manage to do so these structures provide little value in designing the actual system. If you have ever found yourself jumping through near poetic hoops to describe the function of something as simple as a button without being allowed to call it a button, you will be glad to leave these discussions behind.

The reason for this is quite clear; in practice it is often simply impossible to create separate function and requirement trees [4]. For example, the function 'measure temperature' might contribute to the functions 'control position' and 'control temperature'. Consequently, it is impossible to assign a single 'parent function' to the function 'measure temperature'.
This also holds for requirements.

Moreover, many functions and requirements originate from design decisions rather than other functions and requirements. For example, if one decides that an actuator is to be a hydraulic actuator, the functions 'filter oil', 'store oil', 'pump oil' and requirements with respect to the type of oil appear. While choosing a spindle actuator would yield a completely different set of functions and requirements.

Hence, ESL only requires one to define the product decomposition (= PBS). All functions and requirements are formulated in terms of flows between PBS elements and properties of PBS elements. Dependencies between the functions, requirements, flows, properties and PBS elements are automatically derived by the ESL compiler.

AXIOMATIC DESIGN

Axiomatic Design (AD) is a systems engineering methodology that enables the creation of good system designs by improving the causality of system functionality, the system's physical realisation, and manufacturing processes. This is done by using mathematical principles that describe independence between these elements. AD contributes to a better understanding of main and alternative design options. Its applications extend across various industries, from mechanical engineering to (bio)medical systems and even social systems. AD is particularly suitable for addressing problems in high-tech systems development. This article outlines what AD is and how it can be applied.

ERIK PUIK AND RIK LAFEBER

Introduction

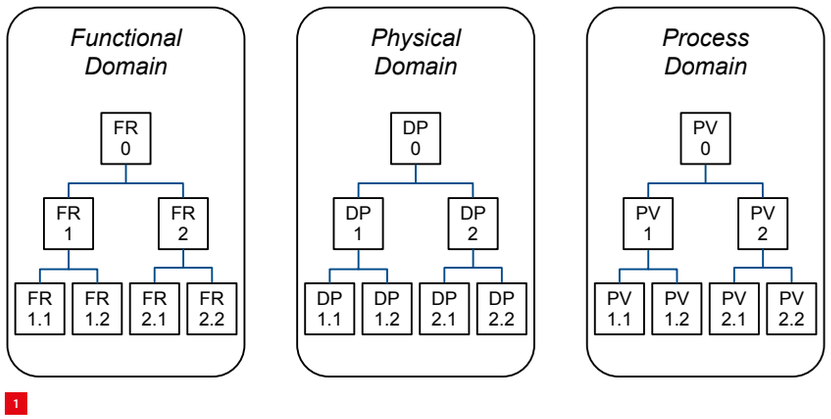
Axiomatic Design (AD) is a concept developed to address the challenges that arise during the design phase in the development of complex systems. The method was developed by Nam P. Suh of the Massachusetts Institute of Technology (MIT) in the second half of the 1970s [1]. AD declares 'Axioms' that cannot be proved or deduced from physical phenomena, which gives the method its name. Initially, a number of six design Axioms were defined. Two of the Axioms have stood the test of time, the others appeared to be corollaries of these two. Since then, the methodology has centred around two primary axioms:

- The first axiom, the 'Independence Axiom', directs the designer to ensure that the functional requirements are independent. This means that each functional requirement should be determined, commonly referred to as 'satisfied', by a specific design parameter and not influenced by others. The advantage of such independence is that changes to one design parameter should not interfere with multiple other functions. Consequently, the design is more robust, easier to control and improve, and less prone to unexpected outcomes.
- The second axiom, the 'Information axiom', encourages minimising the information content of the design. Essentially, the design should be as simple and clear as possible. As few steps or parameters as possible should be required to move from function to physical solution.

AUTHORS' NOTE

Erik Puik is professor of Smart Manufacturing at Fontys University of Applied Sciences in Eindhoven (NL). Rik Lafeber is researcher Microsystem Technology and lecturer Mechanical Engineering at HU University of Applied Sciences in Utrecht (NL).

erik.puik@fontys.nl
www.fontys.nl
www.hu.nl



1

Axiomatic Domains and their hierarchical organisation.

The application of the Axioms contributes to a structured design process, reduces complexity and promotes robustness. It provides a rational basis for design choices and improves communication between team members using the formalised methodology. This introduction to AD focuses on how system requirements are decomposed in AD. To do this, the Independence Axion is applied. The Information Axiom will be described in a future article.

Organizing Domains

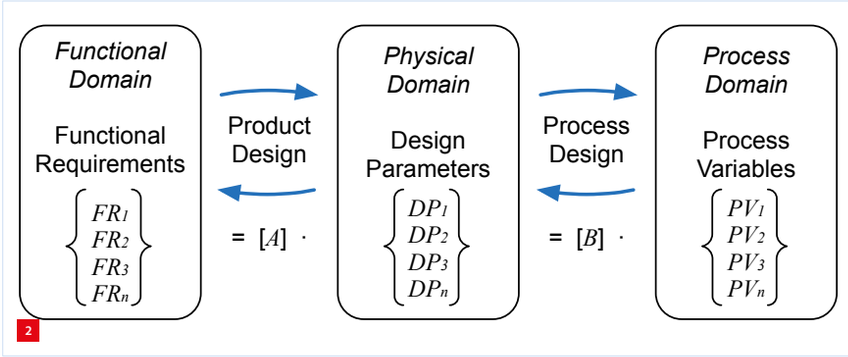
AD provides a systematic method of translating functional needs into functional design, reducing reliance on intuition or guesswork in the design process. This systematic approach creates a clear roadmap, starting with identifying functional requirements, fulfilling these requirements through design parameters, and finally representing these parameters in process variables.

AD demands clear formulation of design objectives through the establishment of 'Domains':

- functional domains, containing the Functional Requirements, from now on to be called FRs;
- physical domains, containing the Design Parameters, or DPs;
- process domains, containing the Process Variables, or PVs.

The domains are hierarchically decomposed as shown in Figure 1.

According to the definition in AD, the Independence Axiom advises to "Maintain the independence of the functional requirements". AD also explains how this can be done from a mathematical perspective, as shown in Figure 2.



Axiomatic Domains and their relations.

The domains, in which functional requirements (FRs), design parameters (DPs), and process variables (PVs) are represented as vectors, are interrelated with design matrices. The design equations according to good AD practice are defined as:

$$\{FR\} = [A] \cdot \{DP\}$$
$$\{DP\} = [B] \cdot \{PV\}$$

Here, $[A]$ and $[B]$ are the product and process design matrices, respectively. If a product design has three FRs, DPs, and PVs, the product design matrix $[A]$ and the process design matrix $[B]$ have the following form:

$$[A] = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$[B] = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

The design equation for the FRs is then defined as follows:

$$\begin{bmatrix} FR_1 \\ FR_2 \\ FR_3 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \cdot \begin{bmatrix} DP_1 \\ DP_2 \\ DP_3 \end{bmatrix}$$

Here, FR_1 to FR_3 are three functional requirements and DP_1 to DP_3 are three relevant design parameters. In this representation, a '*good design*' would be an 'uncoupled' or a 'decoupled' one if the matrix is diagonal or triangular, respectively, as shown below:

$$[A] = \begin{bmatrix} X & 0 & 0 \\ 0 & X & 0 \\ 0 & 0 & X \end{bmatrix} \quad \textit{(uncoupled)}$$

$$[A] = \begin{bmatrix} X & 0 & 0 \\ X & X & 0 \\ X & X & X \end{bmatrix} \quad \textit{(decoupled)}$$

Here, the X-es indicate non-zero elements of the matrix and as such indicate a relation between the associated DPs and the FRs. In an uncoupled design, every FR is related to a single DP, while in a decoupled design, it may be related to more than a single DP, but if the right order is applied to adjust the FRs with the DPs, all FRs can be tuned sequentially. In AD, this design matrix takes a central place because it defines the structure and as such the behaviour of the design.

The design equation for the FRs does not yet include the relation to the manufacturability of the physical system. This is where the process design matrix $[B]$ is applied:

$$\begin{bmatrix} DP_1 \\ DP_2 \\ DP_3 \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} \cdot \begin{bmatrix} PV_1 \\ PV_2 \\ PV_3 \end{bmatrix}$$

The full design equation, which correlates the process variables with the functional requirements, then has this form:

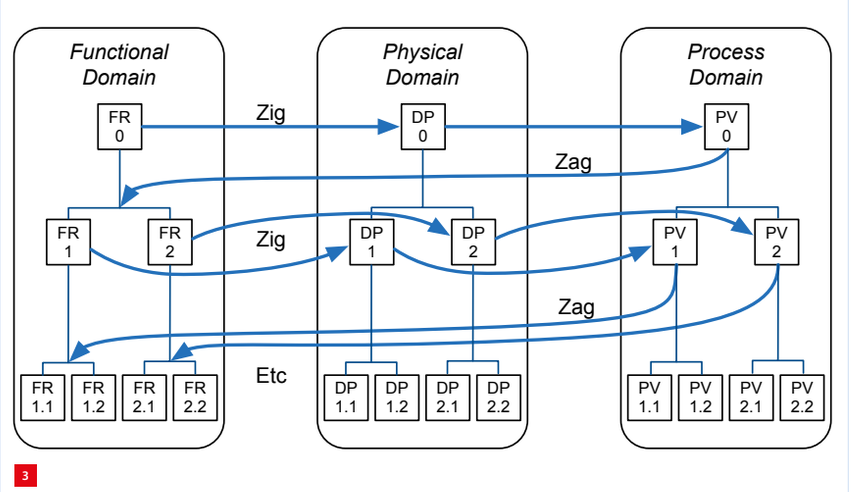
$$\begin{bmatrix} FR_1 \\ FR_2 \\ FR_3 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} \cdot \begin{bmatrix} PV_1 \\ PV_2 \\ PV_3 \end{bmatrix}$$

A caveat must be made here that only a limited number of system designs are sufficiently well understood in practice to successfully apply this equation. The examples in this article are therefore limited to the translation from DPs to FRs.

The process of zigzagging

To check that all FRs are satisfied by their DPs and subsequently the DPs are satisfied by their PVs, AD uses a procedure called 'zigzagging'. Zigzagging is a top-down descent through the design hierarchy, covering all domains sequentially. At each level, it is checked whether the FRs and DPs are satisfied before going down to the next level, as shown in Figure 3.

The process of zigzagging is always performed from the left- to the right-hand side. Zigzagging covers all domains. Successful completion of the zigzagging process will lead to an uncoupled or a decoupled design matrix and satisfies the Independence Axiom, which completes the conceptual



The process of hierarchically zigzagging through the domains.