Part I

Architecture and Systems in the 21st Century

1 Introduction

DOI: 10.1201/9781003213635-1

KEY CONCEPTS

Architecting
Systems
Utility
Conceptual integrity

Over a decade ago when the systems engineering community was rethinking its concepts and practices, an expansive advancement of systems engineering tools and languages began picking up momentum. Dating further back to the mid-1990s, the influence of information technology (IT) has become increasingly significant. Object-oriented languages and tools from the IT community were developed and integrated with systems processes and methods. Over the past decade, this integration has been associated with what is broadly referred to as model-based systems engineering (MBSE). Amongst other things, this was fuelled by and supported the development of large-scale information systems in aerospace and defence such as theatre level mission planning systems in which software, hardware, databases, system operators, and increasingly autonomous vehicles are integrated. The role of system architecting became more central and the need for more formal approaches to system specification, analysis, design, and assurance became central to MBSE. This was echoed in the INCOSE Systems Engineering Vision 2020 expressed in 2007 as noted in the INCOSE *Systems Engineering Handbook* (INCOSE 2015).

Many of the tools and modelling languages of MBSE have been standardised by the Object Management Group (OMG). The established object-oriented language for software development, the Unified Model Language (UML), was adapted and has evolved into the Systems Modeling Language (SysML). Concurrently the standards for software and systems engineering continued to evolve. The need for agreed-upon language is a part of these standards. In the case of the term *system architecture*, driven by the need for agreement, new standards for terminology and concepts were initiated. Given the extent and breadth of activity in the systems engineering community over the past decade, it is important to step back and look at architecting and the engineering of systems in the context of the first two decades of the twenty-first century.

This chapter provides 'brief histories' that serve to distil some of the key activities and concepts and speak to some of the key issues. These derive from the INCOSE *Systems Engineering Handbook* (INCOSE 2015), especially the contribution by Estefan (2007); work that includes a critical analysis of MBSE (Dickerson and Mavris 2013), and summative research for using mathematical model theory in architecture definition and design (Dickerson et al. 2021). One purpose of the summative research was to offer definitions of *architecture* and *system* that are complemented by a mathematical interpretation that can be: (i) used for specification of a technical process for architecture definition and (ii) exploited in engineering practice and relevant standards. The details of the process and mathematically based methods for its implementation are subjects of Chapters 3 and 4 of this book.

What should be clear from the history that follows is that MBSE is far from being in a final stage of maturity. Amongst the areas of work to be finished include a formal basis for MBSE (both logical and scientific) and a stronger synthesis between systems and software engineering.

1.1 A BRIEF HISTORY OF ARCHITECTURE

Architecture is key to the modern practice of engineering. The term would be understood by a general audience as a property of buildings or large-scale structures. In civil engineering, it relates a building's purpose (function), form (how its spaces are organised to achieve the purpose), and construction (what it is built from and how it is built). Although understanding architecture in this way is intuitive and useful, it lacks the precision needed for application to engineering problems. Beyond the design and construction of buildings, architectural ideas are also prevalent in disciplines such as software and systems engineering, management science, and biology.

Despite an extensive common ground, there has been no consensus on the terminology or meaning of architecture across the disciplines. In many ways, a precise practical definition has been elusive. Achieving a precise agreed-upon definition by means of a formal approach could bring unity. The International Organisation of Standards (ISO) established the working group JTC1/SC7/WG42 on System Architecture for this purpose. In the standard for Architecture Description, ISO/IEC/IEEE 42010:2011, a definition was adopted that has been subsumed into later standards (ISO 2011). In 2018, the working group began a routine review of the standard, which is still ongoing at the time of writing of this book. In a shift from a narrow system orientation, recent efforts within have also applied the term architecture to entities not normally considered to be systems.

The early attempts to standardise ideas about the meaning of architecture within systems engineering were in connection with the lifecycle processes standard, ISO/IEC 15288. The first published version (ISO 2002) was influenced by legacy software engineering standards, which conditioned the way in which architecture was conceived. It also combined architecture with design into a single process called Architectural Design. This was separated into two processes in the 2015 update, ISO/IEC/IEEE 15288:2015 (ISO 2015), which is generally accepted as the *de facto* standard for system life cycle development. It has also sought to incorporate the influences and standards from software engineering.

As noted by Wilkinson in Dickerson et al. (2021), the influences of the software discipline date back to the early 1990s, when the IEEE considered architecture to be "the organizational structure of a system or component". This provided a formulation for architecture as a system property, described in terms of static structure (elements and their relations). In 1996, with the aim of using an architectural metaphor as a foundation for systems engineering, Hilliard, Rice and Schwarm (1996) defined architecture as "the highest level conception of a system in its environment". This has several important ideas: architecture is 'high level', includes an 'external focus', and is a 'conception' in the imagination. The ideas described by Hilliard et al. were carried over into the definitions used in IEEE 1471:2000 (IEEE 2000), which defined architecture as "the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment and the principles guiding its design and evolution". The same wording was used in the 2008 issue of ISO/IEC/IEEE 15288. The idea that architecture is subjective is fully embraced in these standards, which introduce architectural descriptions (i.e., models of architecture) as the vehicle for conveying information about the subjective conception of architecture. However, the role of models of architecture for communication has caused a widespread perception that architecture and model are equivalent: Wasson (2005), for example, considers architecture as a graphical model (or representation). By contrast, the popular term 'architecture description' can be understood as referring to a description of a system, just as the term 'blueprint' in civil engineering refers to a blueprint of a building and not of the architecture.

Over the same two-decade period, the Object Management Group™ (OMG™) has used architecture in standards for software development. For example, the Model Driven Architecture® (MDA®) is an approach to software design, development, and implementation. MDA provides guidelines for structuring software specifications that are expressed as models (OMG 2014). In MDA, the software architecture of an application is the basis for deploying the computer code. It provides a comprehensive collection of models that can be used to specify both the software and the computing platform. In the MDA approach, software development focuses first on the functionality and behaviour of a system by means of a platform-independent model (PIM) that separates business and application logic from underlying platform technology. In this way, functionality and behaviour are modelled once and only once. The PIM is implemented by one or more platform-specific models (PSMs) and sets of interface definitions until the specification of the system implementation is complete. A PSM combines the specifications in the PIM with details about how a system uses a particular platform type, but not the details necessary to implement the system.

Despite the fact that MDA has a two decade long standing as an international OMG standard and that the OMG has played a substantial role in MBSE through the development of tools and modelling languages, the concepts of MDA have not played a significant role in the ISO standards related to architecture and systems engineering. This possibly could reflect what appears to be an emphasis on the hardware aspects of systems engineering over those of software in the development of MBSE concepts and practices over the past decade. Another factor might be the lack of a process for specifying the software architecture of modern systems concurrently with the hardware architecture from a comprehensive unified architecture of the system that is attractive to software developers.

1.2 A BRIEF HISTORY OF MODEL-BASED SYSTEMS ENGINEERING

The formalisation of system concepts and processes, and the benefits of so doing have been long recognised in, for example: (i) the foundational work of the biologist Karl L. von Bertalanffy (1967), which was inclusive of many mathematical expressions of systems concepts; (ii) Wymore's codification of *Model-based Systems Engineering* (Wymore 1993), which expressed a programme for systems engineering; and (iii) the mathematical biologist Robert Rosen (1993), who understood the limitations of models and was perhaps the first to recognise the possibility of using Category Theory for systems and scientific problem-solving. Dickerson has also long recognised and investigated formal approaches. The first academic publication with regard to systems concepts (Dickerson 2008) offered an interpretation of the definition of *system* adopted in the standards as a Hamiltonian system in physics to demonstrate that fundamental system concepts could be put on a logical and scientific footing.

1.2.1 EARLY HISTORY OF MBSE

The early history of modelling in systems engineering and key contributions to the formalisation and modelling for software and systems trace back to the second half of the twentieth century. The mathematical concept that a model can be regarded as a nonempty set upon which relations are defined can be traced back as early as Tarski (1954; 1955). A system has also been regarded as a collection of objects with attributes and relations between the objects as well as relations between the attributes and between the relations. Lin (1999) compiled an extensive survey of the mathematical concepts and literature. Lin and Ma (1987) defined a general system to be an ordered pair (M, R) where Mis a set and Ris a collection of relations on M, i.e., a relational structure. However, regarding a system as a relational structure invites confusion between the concept of a system and that of a first-order model, as defined mathematically by Tarski. Their definition might be suitable for the concept of a system that was later expressed, e.g., in ISO 15288 (2015) as a combination of interacting elements. However, it does not formalise the concept that a system is a "whole" consisting of "interrelated parts", which was expressed by Bertalanffy (1967). This issue is a subject of concern in the book. A resolution and deeper explanation of the issue is rooted in the critical analysis of the definitions of key terminology in Chapter 3. A final logical model of the concepts of architecture and system is offered in the summary at the end of Chapter 4.

There is also a significant distinction to be made between the concept of interrelated elements and that of interacting elements. Interaction is very much a concept from physics whereas interrelationship is a concept more common to domains of knowledge such as biology and economics. Furthermore, the mathematical concept of relation as used in first-order model theory *does not per se* address relations between relations, which is a fundamental conceptualisation of interrelationship. Bertalanffy persistently sought a formalisation of the concept of interrelationship but never found one to his satisfaction.

With the emergence of digital computers during this period, Yourdon (1989) introduced a model-based approach for software development from a behavioural viewpoint. He also introduced a graphical modelling language called Data Flow Diagrams to support his approach. His concept of Structured Design was based on a principle that systems should be comprised of modules, each of which is highly cohesive but collectively are loosely coupled. The strongest form of cohesion is based on functionality. The idea was that cohesion within modules minimises interactions between elements not in the same module; thus, minimising the number of connections and the amount of coupling between modules.

Yourdon also introduced the concept of Structured Analysis, which is based on a principle that the specification of the problem should be separated from that of the solution. He proposed that this separation be accomplished by using two types of models: an Essential Model, which is an implementation free representation of the system, and an Implementation Model, which is a specific representation of the software and hardware needed to realise the system. These two types of models are linked by interpretation of a behavioural model into one for implementation. Structured Analysis requires that the concerns of the two types of models be separated. These concepts for using models in software development became the basis for the OMG MDA approach. The concepts and models of Structured Analysis and Design will be explored further in Chapter 4 then carried forward throughout this book.

Wayne Wymore (1993) was one of the early pioneers in the domain of model-based systems engineering. He had a behavioural viewpoint on systems in which the model of behaviour is comprised of the name of the system, its states, the inputs and outputs of those states, and state transitions. A readout function was also provided for outputs of the transitions. Wymore advocated that system design is the development of a model on the basis of which a real system can be built, developed, or deployed that specifies all the requirements using a mathematically based system design language.

Wymore also had a concept of homomorphism between system models as a mapping of the states of the systems, to include their inputs and outputs, in such a way that the two homomorphic models exhibit the same behaviour under the mapping. Homomorphism between functional and implementation system models was intended to assure system intended behaviour. This is similar to Yourdon's concept of Structured Design that the solution should reflect the inherent structure of the problem.

Wymore's concept of homomorphism is generalised in Lin (1999) by the concept of a general system using an algebraic definition. Specifically, given two general system models $S1=M1, R1$ and $S2=M2, R2$, a mapping $h:M1\rightarrow M2$ is a (relational) homomorphism if for each relation $r\in R1, h(r)\in R2$. Note that this definition preserves relations at a structural level of description. It should also be noted that (i) in abstract algebra the mapping h is a function and is not permitted to make multi-valued assignments into $S2$, and (ii) as noted previously, this risks confusion between the concept of a system and that of a model. The mathematical details of this concept will be clarified in Chapter 2 (Logical and Scientific Approach), then used extensively through the book.

Towards the end of the twentieth century, Klir (1991) complemented the concept of a general system with a general systems methodology. Simply stated, he regarded problem solving in general to rest upon a principle of alternatively using abstraction and interpretation to solve a problem. He considered that this could be used both for system inquiry (e.g., the modelling of an aspect of reality) and for system design (i.e., the modelling of purposeful human-made objects). The Klir concept of system inquiry can also be regarded as an approach to architecture and system description as considered in the twenty-first century standards. This will be exploited extensively in the book beginning with Chapter 4 when a relation framework is introduced that implements the concepts of homomorphism expressed by Wymore and Lin.

Looking back on the second half of the past century, key historical contributions were made to establish model-based approaches for system description, analysis, and design. The concepts were strongly influenced by software engineering but were multidisciplinary and sought to establish systems as a domain of knowledge founded on mathematics and science. Generally, a system was regarded as a collection of objects (entities) with attributes and relations between the objects, relations between their attributes, as well as interrelations. A mathematical formalisation of system modelling was consistently sought.

In principle, the formalisation could have been accomplished using the first-order model theory of Tarski and the homomorphism of relational structures prescribed by Klir and Lin. Each of these model-based approaches in the early history of MBSE incorporated a mathematical form of model transformation that preserves structure and behaviour. However, the systems community never achieved closure at this level of formalisation. In the current practice of MBSE over the past decade, a less formal and more intuitive approach has been followed in software and systems engineering using graphical object-oriented modelling languages such as UML and SysML. The idea of transformation between models has been part of this approach but has never been formalised and applied with a precision sufficient for engineering applications.

1.2.2 RECENT HISTORY OF MBSE

Developed in the 1990s by the Chesapeake Chapter of INCOSE with significant aerospace involvement, the Object-Oriented Systems Engineering Methodology (OOSEM) is a top-down hybrid approach that leverages object-oriented software and system techniques. It is model-based and can be implemented in SysML. The core tenet is integrated product development using a recursive Systems Engineering Vee approach. Key system development activities in the methodology include: Analysis of Stakeholder Needs, Definition of System Requirements, Definition of the Logical Architecture, Synthesis of Candidate Allocated Architectures, and Optimisation and Evaluation of Alternatives. OOSEM is supported by a number of tools that have been commercially developed using OMG standards and SysML.

This and other methodologies were reviewed by Estefan (2007) in an INCOSE report. One was based on concurrent systems engineering activities that reflect the Systems Engineering Vee: the Vitech MBSE Methodology. The key activities are: Source Requirements Analysis, Functional Behaviour Analysis, Architecture Synthesis, and Design Validation and Verification. The methodology uses a layered approach to system design (the ‘Onion Model’) and a common System Design Repository. It specifies a System Definition Language based on entities, relationships, and attributes. As noted by Estefan, the methodology is commercially supported by the Vitech CORE product suite.

Another methodology is the State Analysis that was developed by the California Institute of Technology Jet Propulsion Laboratory (JPL) with deep space missions in mind. The methodology uses Goal-directed Operations Engineering and leverages model- and state-based control architecture. States describe the ‘condition’ of an evolving system, such as a spacecraft over possibly long periods of a mission. This is an iterative process for state discovery and modelling. Models are used to describe system evolution. Core tenants include state-based behavioural modelling and state-based software design. The methodology seeks to reduce gaps in the software implementation of systems engineering requirements. The JPL State Analysis can augment the Vitech CORE functional analysis schema to synthesise functional and state analysis.

A formal modelling methodology was developed by Dori (2016): The Object Process Methodology (OPM), which is based on the premise that everything is either an object or a process. Objects exist or have the potential for existence. Processes are patterns for the transformation of objects. States are situations that objects can be in. OPM combines formal Object-Process Diagrams (OPDs) with an Object-Process Language (OPL). OPD constructs have semantically equivalent OPL sentences. The reader should note the similarity with Tarski model theory, i.e., language is interpreted into structures (diagrams in this case).

OPL is oriented towards humans as well as machines. There are also system structural links (relations) and procedural links (behaviour). Structural links are similar to UML class relationships (e.g., generalisation). Three mechanisms are used for modelling: (i) unfolding /folding that refines/abstracts structural hierarchy, (ii) zooming out/zooming that exposes/hides details of an object or process, and (iii) expressing/suppressing: exposes/hides details of a state.

Estefan also reported on two IBM methodologies: The Rational Telelogic Harmony method and the Rational Unified Process (RUP) for systems engineering, which extends development methods for software to methods for system development. Modelling is supported at the levels of context, analysis, design, and implementation by the IBM Rhapsody Suite.

The Estefan survey is representative of the progress in MBSE over the past decade but is not exhaustive. What is evident though from the recent history is that both a functional orientation and an object orientation have been carried forward that are embodied in processes, methods, and languages that are supported by commercially available tools. Formal methods have also progressed. However, the vision of the twentieth-century pioneers remains to be completely fulfilled by a comprehensive approach with a logical and scientific basis that expresses the core concepts of MBSE.

1.3 INVENTION AND UTILITY

Thomas Edison was a noteworthy inventor in the US during the early twentieth century. Remembered for inventing the electric light bulb, he became a wealthy and recognised technology leader of his time. His pursuit of invention could be considered the pursuit of valuable intellectual capital. What was the secret to his success and how did he discover it?

At the age of 22, Edison received his first patent: an electronic vote recorder to be used by legislative bodies. It was a simple concept but very advanced for its time. Votes could be recorded by the flip of a switch, but legislators did not trust this system and preferred to cast their votes by voice. Edison could not sell the invention. The inventor's response was:

“Anything that won’t sell, I don’t want to invent. Its sale is a proof of utility, and utility is success.”

In the years to come, he applied this principle to his engineering practice of invention. He became very wealthy and some would say that he gave up his passion for technology in the pursuit of money. But another view of Thomas Edison might consider that he valued the utility of his inventions and focused on the utility of the technologies emerging in his day rather than on just the technologies. His viewpoint coupled with his passion for technology could not help but make him a very wealthy businessman. Even so, Edison himself said that invention was 99% perspiration and 1% inspiration, and it has been said of him that often he was found late at night sleeping on one of the laboratory benches, in between running experiments.

1.3.1 THE UTILITY OF SYSTEMS ENGINEERING

Engineering was well-established across many scientific disciplines by the beginning of the twentieth century: mechanical, electrical, and chemical, just to name a few. Engineers have since established themselves as the twenty-first century applied scientists and like Edison deliver technical products and services of value. It is a legitimate question to ask how well systems engineers have established themselves as they now enter the third decade of the twenty-first century. The INCOSE *Systems Engineering Handbook* speaks to the value and offers studies and analyses (INCOSE 2015), though many are large-scale and defence-oriented.

There is a general recognition that systems engineering is important for discovering problems early in the development of products, thereby dramatically reducing the cost of scrap and rework in the later stages, or in the worst case, program cancellation. In fact, the leading cause of cancellation for large-scale software-intensive systems development is the failure to capture or manage requirements. Surveys conducted by INCOSE give further insight relevant to MBSE. The results presented by Cloutier and Bone (2015) indicated that most practitioners were primarily performing the technical processes of Requirements Definition and Architecture Definition (as specified in ISO/IEC/IEEE 15288: 2015). This should not come as a surprise. Two decades ago, Hatley's *Process for Systems Architecture and Requirements Engineering* (Hatley, Hruschka and Pirbhai 2000) was focused entirely on these two technical processes as based on a successful commercial track record in the 1990s. The actual business value of MBSE is a longer term proposition in that the cost savings in a shift from document- to model-based management of information nominally is realised only as the product or service enters production and deployment.

1.3.2 THE UPTAKE OF MBSE

There are various sources regarding how well industry as a whole has adopted the processes, tools, and modelling languages of MBSE. In the United States, a recent survey was conducted by the Massachusetts Institute of Technology (Cameron and Adsit 2020). This survey of documentation methods was conducted across several thousand post-graduate students enrolled in online studies at MIT. The survey only reflects the usage of methods at the student's company. It should also be noted that the students were mostly mechanical engineers whose companies did not have an MBSE process in place. It is therefore possible that some or many of the students were enrolled because of corporate interest to explore MBSE possibilities.

Most of the documentation methods reported in the survey were concerned with requirements (slightly fewer than 2,000 responses) and change requests (about 800 responses). Word file usage was the most frequent method in both cases, accounting for about one-third of the responses for requirements documentation, which was comparable to the number of responses that reported using software tools for requirements documentation. Word files accounted for almost two-thirds of the responses reported for change requests. For interface control (about 600 responses), the usage of software tools was less than one-third. Word files accounted for over two-thirds. There were just fewer than 500 responses collectively for UML, SysML, Design Structure Matrix, and OPM; with UML being the most frequent at about one-third of the responses and SysML slightly less.

Reasons for usage were not analysed in the survey. Nonetheless, the scale of preference towards text-based documents for requirements, change requests, and interface control over UML, SysML, and OPM is dramatic. It is also worth noting that UML has not been displaced by SysML in the companies in the survey. This might reflect that software engineers have been using UML for software development for over two decades whereas SysML is more recent; or, on the other hand, that they generally prefer to use UML. The survey is silent on such details.

This is perhaps an appropriate point to reflect on the context and progress in systems engineering that has been made since the first edition of the book, which was written over a decade ago when the community was rethinking systems engineering. There is no definitive research on the benefits and utility of the progress that has been made during the expansive advancement of systems engineering tools and languages over the past decade. This is not to say that progress has not been made. However, research like that of MIT would indicate that an overwhelming impact over the past decade of advancement has not been achieved.

The words of Edison that "utility is success" are perhaps the end goal that the systems engineering community should keep in mind at the beginning of the next decade of advancement that lies ahead.

1.4 THE ROLE OF THE ARCHITECT

Finally, it is worth considering how the system architect fits into this picture. In practice, a system architect has a more user and customer-focused role, whereas a systems engineer might be more concerned with the technical team developing the product or service. In a small-scale project, the functions of the architect and systems engineer may indeed reside in the same person. This is certainly the case in civil engineering. A pioneer of commercial digital computer architecture, F. P. Brooks (1995) succinctly expressed a viewpoint on the role of the architect:

> "*Conceptual integrity is the most important consideration in system design. The architect should be responsible for the conceptual integrity of all aspects of the product perceivable by the user*."

These two statements have been taken as the first and second principles of architecture and systems engineering in this book. This viewpoint is supported by the professional experience of the principal author, tracing as far back to his role as the Director of Architecture for the Chief Engineer of the U.S. Navy (2000–2003) in the Office of the Assistant Secretary of the Navy for Research, Development, and Acquisition. In this role (Dickerson et al. 2003), the architect was positioned between the acquisition authority (the Assistant Secretary) and the government leaders of system development (the System Commands). This is not unlike the position that many system architects in aerospace and commercial practice find themselves in today. This viewpoint was taken in the first edition of this book and it remains unchanged a decade later.

2 Logical and Scientific Approach

DOI: 10.1201/9781003213635-2

KEY CONCEPTS

Conceptual integrity
Knowledge and language
Models in science and engineering
Logic and first-order models

Mathematics is the language of science and engineering. However, neither necessarily uses the formal language upon which mathematics is founded in a formal way. Instead, like many other domains of discourse, their use of language is more descriptive of concepts that have a physical foundation. The validity of these domains is based on repeatable experiments or the successful realisation of concepts in terms of physical objects. Modern mathematics and science enjoy the benefit of thousands of years of human thought, experience, and practice. The professional practice of engineering, spurred on by the industrial revolution, has enjoyed centuries of such benefits. Systems engineering, on the other hand, has enjoyed less than a century of development as a professional practice. As a distinct engineering discipline, from any historical perspective, it must be considered to be still young and maturing.

Systems engineering by its very nature is multidisciplinary. The use of language in this developing domain of discourse is then particularly challenging because it needs a common language to express both domain concepts and system concepts. The technical use of the language of mathematics prevalent in science and engineering has not been developed for systems to date in a way that has been applied across multiple domains (engineering disciplines) to enable a coherent and comprehensive approach to the engineering of both hardware and software for modern products and services, i.e., systems. This chapter begins a discourse on how the language of logic binds mathematics, science, engineering, systems, and software together.

Comparing mathematics and science with the historical and academic disciplines of engineering, it is clear that:

- Mathematics uses

- Formal logic (the predicate calculus of logic) for description
- Logical deduction and mathematical induction for reasoning
- Science uses
  - Mathematics for description and quantification
  - Models and experimental methods for reasoning
- Engineering disciplines use
  - Science and mathematics for description and reasoning
  - Analysis, tests, and prototypes for design and decision
- Software engineering uses languages and tools
  - Computer languages such as the Language C for procedural programming
  - The Unified Modeling Language (UML) for object-oriented programming
  - Computer-Aided Software Engineering (CASE) tools

The separation of (hardware) engineering from software engineering in the above comparison is necessary because these two disciplines are distinct domains of discourse. This distinction raises many issues that are a subject of research. An investigation of the issues, which is accessible to general audience, can be found in the work by Pyster et al. (2015). How architecture and systems can better unify these two domains is a subject of concern throughout this edition of the book.

2.1 EXPERIMENTAL AND LOGICAL BASIS OF SCIENCE

If engineering in general and systems engineering in particular are to be founded upon science, then there needs to be an agreement on what is meant by *science*. The following perspective is offered as a fundamental understanding of the term.

- The Latin word *Scientia* means knowledge.
- The English word *science* refers to any systematic body of knowledge but more commonly refers to one that is based on the scientific method.
- The scientific method consists of
  - Characterisation of observables (by definition and measurement)
  - Formulation of hypotheses (e.g., interpretations of models) which are testable and refutable by comparing:

    – Predictions (based on the hypotheses) and
    – Outcomes of experiments (which must be repeatable)

It will be important to understand the difference between principles and laws in both science and systems. In physics, principles are deeper insights from which laws can be derived. Consider the following:

- Examples of principles

    - The conservation of energy
    - The equivalence of mass and energy
- Examples of laws
    - Newton's Laws of Dynamics
    - The Law of Gravity

The laws of Newtonian dynamics, for example, can be derived from the principle of the conservation of energy. Science generally seeks to develop mathematical models of its laws. Consider Newton's three laws for describing the dynamics of bodies of matter.

- Law of Inertia: *An object will stay at rest or continue at a constant velocity unless acted upon by an external force*.
- Law of Acceleration: *The net force on an object is equal to the mass of the object multiplied by its acceleration*.
- Law of Reciprocal Actions: *Every action has an equal and opposite reaction*.

These laws are given in natural language but can be modelled using algebra and calculus. The power of Newton's calculus allowed many physical systems to be modelled and their behaviour predicted. The simple Law of Acceleration, when properly modelled with Newton's calculus, leads to Hamiltonian mechanics where systems of large numbers of particles or other physical objects are described by partial differential equations. The Hamiltonian classical theory of mechanics developed in the nineteenth century is surprisingly universal. It can be used, for example, to formulate models in quantum mechanics, a domain of physics developed in the twentieth century that dispelled many classical concepts.

If this is sounding too easy, you're right! What makes it hard is that a *problem* is easier to model than finding a *solution*! For example, the well-known equations for a system of three bodies of mass remain unsolved even today. The orbits of the planets in our solar system would be unpredictable if it were not for the dominant mass of the sun!

So, the maturing discipline of systems engineering has something to learn from the historical disciplines of science. The power of formal languages in modelling is just one. But a new level of precision for the description of systems engineering problems will not be a *panacea*. Hard problems and unsolvable problems will remain. Reliable prediction of behaviour is not always attainable.

2.2 SCIENTIFIC BASIS OF ENGINEERING

If systems engineering is to be properly understood, then it must be understood in the general context of engineering. The following definition of engineering will be used for this book:

> *Engineering is a practice of concept realisation in which relations between structure and functionality are modelled using the laws of science for the purpose of solving a problem or exploiting an opportunity.*

The fundamental nature of this definition will become clear throughout its use in the book. Although there are no doubt many other useful definitions of this common term, this definition is well suited to the architecture and systems engineering approach that will be taken.

2.2.1 STRUCTURAL ENGINEERING EXAMPLE OF USING SCIENCE

A simple example from structural engineering can be used to illustrate the relation between science and engineering:

- Problem: how to best arrange bricks to build a bridge
- Purpose: the bridge enables transport across a gap in the terrain
- The underlying physics
  - Law of Reciprocal Actions
  - Principle of Dispersion of Force in a Structure
- Implications for the engineering of the bridge
  - The bridge must transfer sufficiently large forces from the plane of transport to the base of the bridge on the ground.
  - If the bricks are arranged in an arch, then the forces can be evenly dispersed through the structure and transferred to the base.
  - The layers of bricks should be staggered to avoid vertical seams in the mortar. (Such seams would provide paths of stress in the structure through the weaker material that comprises the mortar.)
  - Among the benefits of the design is that the number of bricks required is greatly reduced and the resulting structure can be visually attractive.

It should also be noted that the size of the bridge might be limited by the size of the bricks.

Civilisations have been building bridges such as this for thousands of years without the benefit of modern science. Only a rudimentary understanding of what makes a structure stable and durable was needed. The arch is a *type of structure* that has long been recognised as a preferred arrangement of construction materials (stones, bricks, etc.). Historically the designer might have been an *architect* who would work out various details as to how to build the bridge using this type of structure in a way that best achieved the purpose, i.e., to enable transport across a gap in the terrain and have other properties such as durability and attractiveness.

This short intuitive description of engineering should not lull the reader into a false sense that good engineering comes easily. Notwithstanding the commercial and other contextual problems that can challenge the engineer, reliably understanding and predicting the relationship between functionality and physical structure is a serious technical problem. The laws of science can work both for engineers as well as against them. Murphy's Law is always at work.

The three-body problem in physics is not an isolated example of an inability to predict behaviour. This inability can be complemented by a lack of understanding of the environment of a system. The catastrophic collapse of the Tacoma Narrows Bridge in 1940 stands as one of the great examples. Harmonic oscillations that are well understood in science and engineering were the cause of its collapse due to a lack of understanding of the wind flows in the locale of the bridge. More recently, the Millennium Bridge in London suffered a similar design flaw, which fortunately was corrected before a catastrophic event occurred (but only after the bridge went into service). The unforeseen harmonics, in this case, were caused by large numbers of pedestrians on the bridge who had self-synchronised their cadence. Thus, the use of science in building modern bridges becomes a necessity rather than simply an interesting insight into the behaviour of bridges.

2.2.2 EXAMPLE OF USING SCIENCE IN SYSTEMS ENGINEERING

A more detailed technical example is provided by twenty-first century automotive engineering. The problem in this example will be how to control engine emissions while simultaneously providing cruise control functionality. Advanced driver assistance systems (ADAS) are a class of information-intensive systems that can be developed using science and engineering models to produce solutions that improve driver comfort and safety. A legacy ADAS functionality that has evolved for decades is realised in cruise control systems. In its most basic form, the system maintains a constant speed for the vehicle but gives the driver an option to accelerate the vehicle any time by simply pressing the accelerator pedal. This example will illustrate the relation between science and engineering as follows:

- Problem: how best to limit internal combustion engine emissions during cruise control
- Purpose: maintain driver comfort and control whilst complying with emissions standards
- The underlying science:
  - Law of Inertia
  - Law of Acceleration
  - The chemistry of internal combustion engines
- Implications for the engineering of the ADAS and vehicle engine:
  - A mapping is needed of engine states versus emissions
  - Operating ranges within the permissible states must be robust enough to
    – permit the ADAS to maintain a constant vehicle speed and
    – meet driver demands for vehicle acceleration

One particular challenge in this class of problems is that despite the advances of a century of study of the chemistry of internal combustion engines, it is currently not possible to model or simulate engine behaviours at a level of fidelity that can be used to reliably predict compliance with emissions standards. Measured data from an engine is the basis for creating a reference model that is referred to as an engine map.

A coordinated control architecture design (Lin et al. 2018) can be applied to this class of emissions control problems. The concept is that the existing architecture of the ADAS control of the engine will not be altered but instead a separate emissions control function that acts as a governor will be interfaced with the existing cruise control function and the engine control unit. One advantage of this approach is that the modular system architecture of the integrated solution will not require redesign or extensive retesting of the existing systems.

For the sake of simplicity of illustration, only two design objective constraints will be specified for emissions and only two state variables for the engine. The actual problem has dozens of variables and constraints. This problem was a subject of research in a 5-year programme sponsored by the UK Research Councils as summarised by Dickerson and Ji (2018), the results of which led to a patent on emissions control (Dickerson, Ji and Battersby 2018) amongst other advancements.

In this simplified problem, the two objectives are $z1= CO2$ emission measured in kg/hour and $z2=CO$ emission measured in kg/hour. The constraints are $z1\leq30$ and $z2\leq0.1$. The state variables will be engine speed, $x1$ in revolutions per minute (RPM) and engine torque, $x2$ in Newton-meters (Nm). An engine map has been measured for the state space $1600\leq x1\leq2000$ and $100\leq x2\leq240$. As a mathematical mapping, the map is defined by two response surfaces $fx1, x2= z1$ and $gx1, x2= z2$. However, as already noted, the actual engine map is a table of discrete measurements. The selection of measurement points for creating the map is guided by an understanding of the chemistry of internal combustion engines within the intended operating ranges (states) of the engine.

The constrained operating space of the engine displayed by the shaded region of the graph in Figure 2.1(c) depicts the transformation of the constraints on the objectives into engine torque and engine speed through the inverse mappings of the two response surfaces that are displayed in Figure 2.1(a) and (b), respectively. The existing ADAS cruise-control functionality provides torque demands to the engine to maintain a constant vehicle speed (which has been set by the driver). In a fixed gear, this corresponds to a constant engine speed, e.g., 1600 RPM. In this case, the graph in Figure 2.1(c) indicates that the existing ADAS functionality could make torque demands on the engine over the full operating range (100–240 Nm) without violating the emissions constraints. The control of the engine speed by the ADAS would be sufficient for maintaining an appropriate vehicle speed (corresponding to engine speeds near 1600 RPM) and emissions.

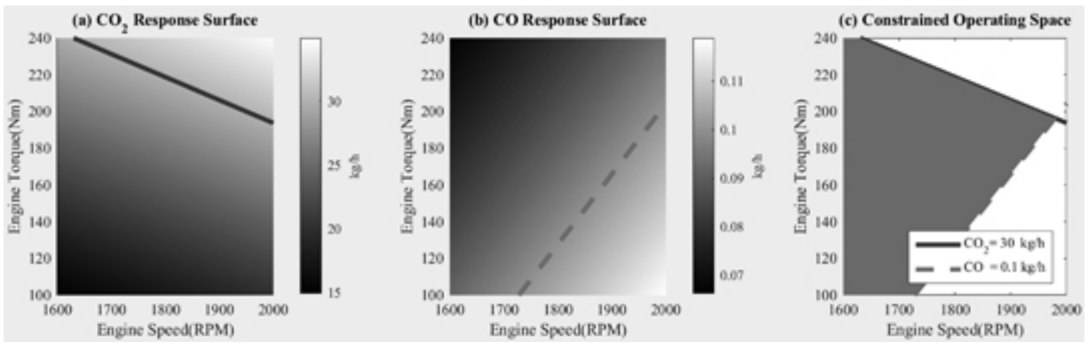

FIGURE 2.1 Engine emissions response surface model and constraints: (a) CO_2 response surface, (b) CO response surface, and (c) constrained operating space.

At a higher engine speed, e.g., 1700 RPM, the torque demand would need to be limited to about 230 Nm. Thus, the new functionality is indeed acting as a governor on the torque demand permitted by the ADAS. At engine speeds of more than 1700 RPM, the torque demand will need to be not just limited by an upper bound, but it will also need to stay above a lower limit. For example, at 1800 RPM torque demand would need to be limited to be between 130 and 220 Nm. If a torque less than 130 Nm is needed to maintain the vehicle speed that corresponds to 1800 RPM, then the ADAS or the driver will need to shift gears to lower the engine speed. In general, on a smooth level road surface in light traffic conditions, due to inertia, the ADAS can maintain a constant vehicle speed by making small demands for torque to overcome friction and wind resistance. If the vehicle is travelling over hills, a larger amount of torque will need to be demanded.

The governor function for permitting the driver to accelerate from a state of cruising is similar but more complicated. At the start of acceleration action from a constant speed cruise, the engine is initially at a given state (RPM, torque) within the constrained operating space of the engine displayed by the shaded region of the graph in Figure 2.1(c). When the demand for increased torque is sent to the engine, the result will be increased RPM, i.e., acceleration. Keeping the engine state within the permissible range of states during the acceleration can again be accomplished by limiting the torque demand but the engine speed must also be limited to a permissible range. Thus, the governor must be able to limit the engine speed as well as the torque demand. This could be accomplished by reducing the torque demand permitted to stop the acceleration. For example, if the driver began accelerating from the engine state of 1600 RPM engine speed at 170 Nm, the torque demand would need to be limited to about 205 Nm and when the acceleration brought the engine close to a speed of 1900 RPM, the torque demand would need to be reduced to a level that stops the acceleration. This point is determined by the CO_2 constraint in the graph in Figure 2.1(c). As with the governance of the constant speed cruise, it may also be necessary to shift gears to keep the engine within the permissible range of states.

The analysis of the constrained operating space to specify operating ranges within permissible engine states provides a straightforward model-based method to integrate an emissions control governor with the ADAS. The implementation of this method will require the specification of real-time software. The governor example will be completed in Chapter 4 using Structured Design.

2.3 LANGUAGE, LOGIC, AND MODELS

If the laws and models of science are to be used as a basis of precision and repeatability in engineering, then criteria for what constitutes a good model need to be considered. For architecture and systems engineering, models for both science and conceptual solutions must be considered.

- Stephen Hawking (1988) attributed a good model in physics to be
  - Simple
  - Mathematically correct
  - Experimentally verifiable
- By analogy, a good logical model of a concept should be
  - Simple
  - Logically well-formed and consistent
  - Verifiable through logical interpretation

Concepts can be expressed in various ways, but in practice, natural language sentences remain the most common way to express concepts. The modelling of concepts in this book is motivated by the mathematical theory of first-order models using the predicate calculus. See, e.g., *Models* *and Ultraproducts* (Bell and Slomson 1969). The terms *model* and *sentence* have precise meanings in the predicate calculus. The syntax of sentences in mathematical logic is specified by the propositional calculus.

- The formal languages of mathematical logic are the
  - Propositional calculus
  - Predicate calculus
- The term *calculus* derives from the Latin for calculation.
  - In logic, *Calculus* refers to the calculation of truth.
  - By contrast, in the *Calculus* of Newton and Leibniz, the term refers to the calculation of limits.
- *Propositions* are declarative statements and are represented by propositional variables, i.e., the propositions are represented as abstract variables such as , q, ….
- *Predicates* are statements of relations and are represented by predicate letters such as , Q, …. The variables related by a predicate letter are represented as abstract variables such as , v, …. Thus, P(u, v) represents an abstract relation between two variables.
- *Quantification and Interpretation*: given a domain of knowledge, these symbols are interpreted into the domain, i.e., they take on meaning and values in the domain. When each variable is interpreted into either a single value or a specified range of values, the predicate is said to be quantified.

Propositional logic is also referred to as the Sentence Calculus because it defines the syntax of logical sentences. Just as algebra makes general statements about numbers (e.g., a2+ b2= c2) that can have various interpretations which are true or false (if proper syntax is followed), so too statements in logic are sentences formed of abstract variables that represent decidable assertions that are generally formed from relations. The types of relations of primary interest in systems engineering will be constraints and interactions between elements.

The shaded region in Figure 2.1(c) shows a constraint relation between the two state variables in the engine operating space: engine speed, x1, and engine torque, x2. Each variable has been quantified by its range of values, 1600≤ x1≤ 2000 and 100≤ x2≤ 240. Thus, the abstract predicate P(u, v) can be interpreted as the joint constraint relation in two variables in the figure, and it has been quantified. This is the level of formality that is behind every well-formed mathematical model. In the practice of science and engineering, this formality is suppressed, but if the model does not conform to this formal basis, then it will be logically incorrect and exhibit errors.

This is how first-order model theory is conceived in mathematics. A *sentence* is a well-formed formula of predicates that is fully quantified, i.e., the formula adheres to the syntax of logic and there are no free variables in the formula. Free variables (i.e., variables that have not been given proper scope through quantification) admit unbounded interpretation, which is problematic. In natural language, the scope of terms can be provided by certain types of adjectives, e.g., the speed of an aircraft can be quantified by the adjective subsonic.

For a sentence in the predicate calculus, a *model* (of the sentence) is a relational structure (i.e., a set with a collection of relations on the set) into which the sentence can be interpreted and its validity can be reasoned about. The shaded regions determined by the constraints in Figure 2.1(a) and (b) are an example of a collection of two relations that form this type of structure, i.e., there are two relations in the structure. One is the constraint on engine states from CO_2 regulations and the other is a constraint on the states from CO regulations. They are joined by set intersection (which corresponds to a logical '*and* statement') that forms the constrained space.

The relational structure is said to be a *model of a sentence* if the sentence is valid when interpreted into the structure. This means that the 'image' of the interpretation is 'true'. (The model then represents the sentence.) In natural language, a sentence for Figure 2.1(c) could be as simple as, 'There is an operating space of the engine that is constrained by emissions regulations'. This is intuitive but very ambiguous. In the set-theoretic example given by the graphs in Figure 2.1, the constrained operating space of the engine is a *non-empty set*. If the relations in the figure had an empty intersection, then the statement of constraints would be invalid; it cannot be realised.

This type of model is referred as 'first order' because the predicate calculus is a first-order logic. A general but technical review of the model theory for the non-specialist is given by Wilfrid Hodges in the Stanford Encyclopedia of Philosophy (Hodges 2020). The power of this understanding of models is that it crosses the boundaries of science, natural language, and software engineering. Chapter 3 on concepts, standards, and terminology will conclude with a demonstration of how first-order model theory applies to natural language and the definition of terms. The structured methods of Chapter 4 will show how the theory applies to system specification using modelling languages such as UML.

To the engineer and the scientist, this level of logical detail might seem at best to be an interesting insight into the modelling of systems. Indeed, just as certain details of mathematics are suppressed in science and engineering, many of these logical details should be managed by proper use of natural language and also in the background logic of a modern systems modelling tool using a language such as UML or SysML. However, first-order models are much like the use of science in building modern bridges where the scientific understanding of the behaviour of bridges becomes a necessity rather than simply an interesting insight. Just as bridges can fall down, concepts can fall apart when subjected to the complexity of integrating hardware and software into a holistic system. If conceptual integrity is not maintained, the result is failure to realise the concepts and requirements that the system was intended to fulfil. Indeed, the system might even fail to be built.

3 Concepts, Standards, and Terminology

DOI: 10.1201/9781003213635-3

KEY CONCEPTS

Purpose of engineering standards
Interpretation of terminology
Concepts and model theory
Essential technical processes

The terminology of systems engineering has different meanings depending on the user. Standards organisations seek to normalise terminology across a broad community of users. However, the current standards relevant to systems engineering *do not* provide the precision required by a rigorous model-based approach to architecture and systems engineering. Precise but practical definitions of systems terminology that have collective consistency and conceptual integrity and also have engineering utility are needed for the specification of technical processes and methods for the system and architecture definition of engineering solutions to problems of commercial interest. This chapter introduces precisely defined terminology that can be used to complement the current terminology in accepted standards and to specify processes and methods based on the interpretation of terms.

Beginning with this foundation chapter, a lexicon of terms and definitions is developed that supports the specification of essential technical processes and methods. Their use and utility are demonstrated in the tutorial case study covered by Chapters 5, 6, and 7. A more comprehensive lexicon is provided in Annex A-3. The concepts and terminology used throughout this book are derived primarily from ISO/IEC/IEEE 15288:2015 (ISO 15288) to the extent possible. This is the most recent and most widely adopted international standard on software and systems engineering life cycle processes. This and other standards will be complemented by research findings to establish the authority and correctness of the concepts and terminology in this book.

Based on the first-order model theory of Tarski presented in Chapter 2, the pairing of models with concepts expressed in terms of a language is then both intuitive and mathematically based.

This approach demands both simplicity and rigor. The essential terminology and technical processes presented in this book underlie those specified in the relevant international standards for systems engineering. These are based on recent summative research by the authors and their collaborators (Dickerson et al. 2021). It should be noted that the international standards also consider the commercial and life-cycle aspects of engineering such as definition and management of the system configuration, and translation of system definition into work breakdown structures, amongst others. The essential definitions and processes presented in this book are intended to complement the standards; not to be a replacement.

This chapter concludes with a demonstration that concepts expressed by natural language definitions of terminology can be modelled with the same level of formality as the science-based advanced driver-assistance systems example in Chapter 2. The foundation that has been laid for a scientific basis of engineering using first-order model theory will be further demonstrated by modelling a widely accepted definition of the term system. This will be an important step towards establishing that the approach taken in this book is a qualified approach to model-based systems engineering. Chapter 4 will carry this further by showing how the concepts of architecture and systems can be expressed in a logical model that enables engineering through the methods of structured analysis and design. Using the common thread of an underlying logic that realises the science and engineering concepts of the emissions control governor investigated in Chapter 2, a model-based specification of a system solution will then be described to conclude Part I.

3.1 SYSTEMS ENGINEERING STANDARDS AND ORGANISATIONS

When dealing with the standards promulgated through various national and international organisations, it is important to understand that the standardisation of concepts and terminology is an agreement between users, practitioners, and other stakeholders of the standard. As such these standards tend to reflect a consensus of current best practice across differing business and technical domains. Furthermore, an international standard must also account for interpretation into multiple languages. It can easily be the case that a single key word in English, for example, has no corresponding word in one of more of the languages of Europe, much less the many languages and dialects of countries such as India and China.

With so many influences on the choice and use of terminology, it should *not* be expected that the current concepts and terminology of systems engineering have been formalised to a level that bears the test of logical and scientific precision. In the domain of science and mathematics, the use of precise symbolic languages has greatly alleviated this problem, but it has taken centuries for that normalisation to occur. Systems engineering in comparison with science and mathematics is still a young subject that has a significant intellectual distance to go before its concepts and terminology can be normalised to the level of precision and ubiquity that is currently enjoyed by science and mathematics.

In the current state of standards and practice, the key international organisations with published and broadly adopted concepts relevant to architecture and systems engineering are the

- International Organization for Standardisation (ISO)
- British Standards Institute (BSI)
- International Electronics & Electrical Engineering Association (IEEE)
- Institution of Engineering and Technology (IET)
- International Electrotechnical Commission (IEC)

- Electronics Industries Alliance (EIA)
- Electronics Components Association (ECA)
- Object Management Group (OMG)
- International Council on Systems Engineering (INCOSE)

INCOSE is not a standards organisation but rather is an industry focused professional society founded to develop and disseminate the interdisciplinary principles and practices for systems engineering. It does, however, work closely with standards organisations. For example, the INCOSE *Systems Engineering Handbook*, 4th Edition (INCOSE 2015) was published in parallel with the publication of ISO/IEC/IEEE 15288:2015 (ISO 2015), which is the current *de facto* standard on systems and software engineering. The handbook seeks to elaborate the standard for the purpose of industrial practice.

ISO is the largest developer and publisher of a broad range of international standards. The BSI is a national standards body that was awarded a Royal Charter in 1929 and reaches across the international standards community. The IEEE is a non-profit organisation and is a leading professional association for the advancement of technology, publishing not only standards but also a wide range of technical journals. The IET is a multidisciplinary professional engineering institution that was formed from the Institution of Incorporated Engineers and the Institution of Electrical Engineers, which date back to the late 1800s in the United Kingdom. The IEC is a leading organisation that prepares and publishes international standards for electrical, electronic, and related technologies. The EIA is a trade organisation representing and sponsoring standards for the U.S. high technology community. The ECA is an associate of the EIA and manages EIA standards and technology activities for the electronics industry sector comprising manufacturers and suppliers.

The OMG is an international, not-for-profit computer industry consortium with open membership, which has task forces that develop enterprise integration standards for software related technologies and industries. Being an open group, the OMG operates under a different business model than the previously mentioned organisations. All of OMG's standards are published and accessible without fees on an open website. Unlike professional societies such as ISO, the IEEE, or INCOSE that raise operating funds through fees for the distribution of standards and journals, the OMG raises operating funds by other mechanisms.

The relationship between the systems engineering standards of ISO, the IEC, IEEE, and EIA can be understood by considering the level of detail provided and the breadth of scope across the system life cycle considered. The ISO/IEC/IEEE 15288 standard has the greatest breadth of scope but at a lesser level of detail. The IEEE 1220 standard provides a much greater level of detail but at a reduced breadth of scope. The EIA 632 and EIA/IS 632 standards are in between these ISO/IEC 15288 and IEEE 1220 standards in terms of level of detail and breadth of scope.

3.2 DEFINITIONS OF KEY TERMS

This part of the chapter introduces a half dozen foundational terms and definitions suitable for the model-based practice of System and Architecture Definition that will be introduced in Chapter 4 and demonstrated in the Part II tutorial case study. This core lexicon is not simply a list of terms and definitions. The succinct comparative and critical analysis of the definitions of the terms should provide the reader and student of the book a basic vocabulary and understanding to speak the language of architecture and systems knowledgably.

Commonly accepted definitions will be cited when available but when there is no accepted definition or if there are choices to be made between terms, definitions have been specified or modified in a way that provides a collection of terms that supports a model-based approach to architecture and systems engineering. This gives the collection conceptual integrity. It is appropriate to recall the Principle of Definition originally stated by F. P. Brooks (1995) that was adopted by Dickerson and Mavris (2010) and adapted by Dickerson et al. (2021):

Formal definition of concepts is needed for precision;
prose definition for comprehensibility.

This principle applies to all the fundamental terms of architecture and systems engineering which are defined in this book in terms of natural language because they admit interpretations into the graphical models of Unified Modeling Language (UML) and Systems Modeling Language (SysML). Although these two modelling languages are not formal, the underlying way in which they are used in the book is. Relevant concepts, methods, and applications have been published by the author over the past decade; the foundational research is published by Dickerson (2008; 2013), and by Dickerson and Mavris (2013). The summative research of the authors and their collaborators (Dickerson et al. 2021) offers a critical analysis of the definitions *architecture* and *system* as currently used in the relevant international standards.

A brief summary and critical analysis of foundational terminology for systems and architecture is now offered. A further lexicon of terms is presented in Annex A-3. At the simplest level of abstraction, the concepts of systems and architecture can be organised around those of combinations, partitions, and arrangements. These terms provide a simple intuitive description of a selection of elements joined into a whole, separation of the elements, and the composition of relations of the elements. The following foundational definitions will be used in the essential technical processes and structured methods in Chapter 4, and in the case studies in Sections II and IV:

Structure
Architecture
Model
Functionality
System
Engineering

Definitions of each of these terms follow below in the order above. The 'essential definitions' offered in this chapter are taken from recent summative research (Dickerson et al. 2021). They are subsistent in the sense of being an economical choice of generally understandable words that have a meaningful mathematical interpretation. The essential definitions are intended to complement the definitions from standards and authoritative sources.

3.2.1 COMBINATIONS, PARTITIONS, AND ARRANGEMENTS

A *combination* is a collection of two or more elements selected or identified from a larger set that are in some sense joined together, i.e., combined. For example, a water molecule is a combination of two hydrogen atoms and one oxygen atom. However, no reference is made to relationships between the elements.

A *partition* of a set is a 'covering' of the set by a collection of subsets whose members are pairwise disjoint (i.e., have empty intersection). In mathematics, a covering of a set is a collection of subsets whose union is the set. The collection $\{\{a\}, \{b\}, \{c, d\}\}$ is a partition of the set {a, b, c, d}. A simple example from mathematics is the set of even and odd counting numbers which partition the counting numbers into two disjoint sets.

The term *arrangement* implies a composition of relations of selected elements of a set for a purpose, e.g., an arrangement of furniture in a room or making arrangements for a meeting. In music, the term can refer to a form of musical composition. The term can also have a specialised meaning such as an ordering or permutation in mathematics. However, ordering arrangements do not need to be linear and generally are not. A *scientific arrangement* or method may be defined as the gathering of individual objects into a synthetic whole for an experiment. This concept is very close to that of a system.

3.2.2 STRUCTURE

There are many definitions of the term structure and sometimes structure is confused with arrangement. In a dictionary definition, it is clear that as both a noun and an adjective this term has a stronger meaning than the term ‘arrangement’:

Noun: *The mutual relation of the constituent parts or elements of a whole as determining its peculiar nature or character; make, frame.*

Adjective: *Organized or arranged so as to produce a desired result. Also, loosely, formal, organized, not haphazard.*

Structure in the context of systems is distinguished from ‘arrangement’ in that it refers not just to the system elements but also to system properties.

It is worth noting that structure and organisation have been widely used in engineering but are generally not defined in the standards relevant to architecture and systems. Some of the definitions and usages of the term structure in engineering tend to focus on joining and connection of parts, e.g., in the ontology proposed by Sowa in *Knowledge Representation* (2000). As will be noted for the OMG definition of system architecture (in Section 3.2.4), the parts of a system may need to be isolated as well as connected. The essential definition put forth by Dickerson et al. (2021), on the other hand, considers both joining and separation:

Structure is junction and separation of the objects of a collection defined by a property of the collection or its objects.

This definition is at a higher level of abstraction than currently used in the standards but is readily applicable to physical examples. In civil engineering for example, a building is a collection of objects that includes rooms, which are joined as well as separated to achieve a defined purpose. Buildings in civil engineering are referred to as structures. The essential definition also has a mathematical interpretation, e.g., the property of divisibility by two defines a partition in the counting numbers that separates the even and odd numbers into disjoint subsets. This concept of structure therefore adheres to the Principle of Definition.

3.2.3 SYSTEM ARCHITECTURE

There are various engineering definitions of this term and the unscoped term ‘architecture’ is reported to have at least 130 definitions in various ISO standards. Three key definitions of architecture will be considered, one from the ISO//IEC/IEEE standards, another from OMG, and the other being the essential definition introduced by Dickerson et al. (2021). The definition used in ISO/IEC/IEEE 15288:2015 (ISO 2015) has been adopted from ISO/IEC/IEEE 42010: 2011 (ISO 2011):

> *<System> architecture is the fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution.*

This definition is intended to complement the definition of *system* in the standard (ISO 2015). It lends itself to consideration of how the elements of a system are organised. However, the definition has certain ambiguities that make it difficult at best to apply rigorously to model-based methods. For example, the phrase "embodied in its elements, relationships" has led to debates as to whether architecture is a model. See the summative research (Dickerson et al. 2021) for further details.

3.2.4 ARCHITECTURE: OMG DEFINITION IN MDA 1.0

Amongst the myriad definitions of system architecture is the one from the OMG offered two decades ago (OMG 2003):

> *The architecture of a system is a specification of the parts and connectors of the system and the rules for the interactions of the parts using the connectors*.

The definition is quite practical for understanding the software and physical implementation of a system architecture. However, it is silent on the *separation of the parts*. This is a problem for the civil engineering of a building or the systems engineering of a radar. The system architecture in both cases needs to specify how the elements are 'isolated' from each other. In a building, this could be sound isolation between rooms. In a radar, this could be high-voltage isolation between the transmitter and receiver chains. Thus, the definition could be improved by replacing the term 'connectors' with 'connectors and isolators'.

3.2.5 ARCHITECTURE: ESSENTIAL DEFINITION

The essential definition of architecture is an abstract definition that is more precise and broadly applicable than the previous two. It is independent of any definition of system but can be used to express the relation between Architecture and accepted concepts of System currently used in the international standards. For example, system architecture can be understood as the fundamental form (or structure) of a system. The definition offered by Dickerson et al. (2021) expresses an intimate relation between architecture and structure:

> *Architecture is structural type in conjunction with consistent properties that can be implemented in a class of structure of that type.*

Such properties are said to be compatible with the type and are referred to as *architectural properties*. This language is unavoidably abstract because architecture is an abstraction, e.g., of a building or a system which is concerned with its structure and properties. For example, reliability properties can be implemented in a class of structures of parallel type for certain systems; a simple example being a string of lights (see also Chapter 12). The term *type* is a specialised property that can also be referred to as a *classifier*. It identifies the fundamental structure that is used to express the form of a system. An architecture is then a class of structure that is further defined by architectural properties. It can be thought of as a coupled pairing of properties: (*Structural Type, Architectural Properties*). This definition will provide a basis for specifying system models and their transformations with the precision needed for the rigorous process in Chapter 4.

3.2.6 MODEL

A general audience would understand the term 'model' as a *representation of something* or an excellent example according to a dictionary definition. Familiar examples are scale models, for example of an airplane or an automobile. Children's toys can also be models. Scientists and engineers have used models extensively, both physical and abstract, to understand and reason about real-world systems. In the twenty-first century, models have become increasingly digital even to the point of claims of models as 'digital twins' of physical entities. As excellent as a model might be though, by its very nature it is *not* the 'real thing'. Models are representations of 'referents', i.e., physical or logical entities.

The term model always *refers* to something, i.e., it is a *model of a referent*. Confusion can arise when a model is regarded as *The Model of the referent*. Amongst the anecdotal sayings about modelling and simulation is one that states, "all models are wrong, but some are useful". This issue is explained using simple mathematical models in biological science by Rosen (1993).

In order to avoid confusion between models and architecture, the well-established Tarski definition of 'model' from mathematics and logic is used throughout this book. This was explained in Chapter 2. It can also be used to complement natural language definitions. In mathematics, diagrams or symbolic expressions are generally used to represent models; but formally, models are *interpretations of sentences into structures*. The sentences can be regarded as concepts about the referent.

Tarski model theory (1954; 1955) offers the following simple but formal definition:

> *A model is a relational structure for which the interpretation of a sentence in the Predicate Calculus becomes valid (true).*

This is called a *fully interpreted first-order model; and is the basis for so-called model theoretic truth*. A relational structure in set theory is a set *M* and a collection (i.e., a set) of relations {Ra} on M. In modern mathematics, concepts are realised in sets.

The term *concept* was discussed in Chapter 2. In precise terms, a concept is expressed as a *sentence* that has a *referent*, something that exists physically or abstractly. The terms concept, idea, and thought are used loosely both in general and in various knowledge domains. However, in modern psychology, *thought* is an activity of the mind and its existence can be identified by measurement of electrical activity of the brain. The term *idea* can be regarded as a thought that is suggestive of something, e.g., a purpose or course of action. In philosophy, a *concept* is a thought or idea which corresponds to a distinct entity or class of entities, or to its essential features. In Latin, the term *conceptum* means 'something conceived'.

When a sentence is expressed in the Predicate Calculus, the interpretation into a relational structure is exact. When a sentence is expressed in a natural language, the interpretation will necessarily be subject to the ambiguities of the language. Thus, concepts and terminology have an intimate relation that is central to any practice of model-based systems engineering.

3.2.7 FUNCTIONALITY

A straightforward adaptation of this term taken from dictionary definitions is:

Functionality is the purpose a system is intended to fulfil.

This can be expressed by verb-noun phrases, such as 'track aircraft'. This construct is an implementation free view of a system. Functionality can be used to support a technical process of system definition and the functions within a system. Concept realisation is accomplished in part by system functions.

3.2.8 ENGINEERED SYSTEM

The most widely accepted definition of an (engineered) *system* is given in ISO/IEC/IEEE 15288: 2015 (ISO 2015):

a combination of interacting elements organised to achieve one or more stated purposes.

Automobiles are engineered systems; computers are too. These are just two examples from everyday life. Almost every product or service used in everyday life can be viewed as a system using this definition. A careful examination of the definition shows that it is somewhat biased towards a 'white box' viewpoint of the system, i.e., the system elements. Nonetheless, it has sufficient precision to be applied to model-based processes and will be in Part II.

3.2.9 SYSTEM: ESSENTIAL DEFINITION

The essential definition put forth by Dickerson et al. (2021) is at a higher level of abstraction than the above definition. It can therefore be applied more broadly and also to systems that are not engineered.

A system is a set of interrelated elements that comprise a whole, together with an environment.

A system then is not just a set or combination of elements; rather it is a pairing that is a coupling of two sets of elements: (S, E), where S is a set of elements defined as the system elements and E is a set of elements defined as the environment of the system. This definition will provide a basis for specifying system models and their transformations with the precision needed for the rigorous process in Chapter 4. The more intuitive but less formal term *form* can be used in place of *comprise*. Thus, the system elements can then be said to *form a whole*.

The term System is widely used in science, mathematics, and engineering. A general system can be regarded as a combination of interrelated parts that have properties as a *whole* not possessed by any of the parts alone. Our solar system is a natural system, i.e., an example of a general system occurring in nature. Other physical entities from atoms to animals to the universe itself are systems too; so too are weather systems. The list goes on: business systems and enterprises, economic and political systems, and so forth. It is perhaps only limited by the imagination. The term *interrelated* is more general than and is inclusive of the term *interacting* that is used in the current standards, i.e., interaction can be a type of interrelationship. Abstract systems such as the counting numbers also conform to the essential definition but not to the definition used in standards. Engineered systems are distinguished from other systems by human intent.

3.2.10 ENGINEERING

The nature of modern engineering is based on its evolution in Western civilisation (Finch 1951). Engineering of a product begins with craft but must be honed using the laws of science for precision and repeatability in commercial production. The following definition (Dickerson et al. 2021) aligns well with the essential definitions of structure, architecture, and system that have been proposed:

> *Engineering is a practice of concept realisation in which relations between structure and functionality are modelled using the laws of science for the purpose of solving a problem or exploiting an opportunity*.

It is worth noting the architectural viewpoint in this definition. The association of structure and property (i.e., functionality) conforms to the pairing in the essential definition of architecture.

3.3 MATHEMATICAL INTERPRETATION OF ARCHITECTURE

As noted in the definition of partition (Section 3.2.1), the set of even and odd counting numbers is a partition of the counting numbers into two disjoint sets. The essential definition of structure conforms to this simple mathematical example. The property of 'divisibility of a counting number by the number two' defines the partition. Therefore, the essential definition of structure adheres to the Principle of Definition because it has a meaningful mathematical interpretation. Using an argument similar to the structuring of the counting numbers into disjoint subsets, the essential definition of architecture can also be demonstrated to adhere to the Principle of Definition by specifying a structural type paired with an architectural property. A straightforward example with engineering utility is a *constraint architecture* that defines a feasible region of a design space. The properties of interest are the constraints on the space. The structure will not be as simple as that of a partition. Rather than separating subsets the constraint architecture will be concerned with joining them through set intersection. This example (as well as Chapter 12) is especially useful not just because of its engineering utility but also because it can be used to demonstrate why not all properties can be implemented in a given class of structures.

3.3.1 APPLICATION TO ENGINEERING THE ADAS ECG

The constrained operating space of the diesel engine in Figure 2.1 for the emissions control governor is a typical example. The state variables were each restricted to an interval of values that depended on an initial state. For a single real-valued variable, one type of a such a constraint is in the form $a \le x \le b$, where a and b are distinct. If $a = b$ then the constraint is degenerate and not really an interval. The numbers a, b are referred to as the bounds of the interval, or more loosely as constraints; but mathematically the constraint is a relation between the bounds and the variable. Another type of constraint is in the form $a<x<b$, which is an open interval. In this case, if $a=b$, the interval is the empty set and the constraint is void (and invalid). These two types (which involve only a single variable) provide the simplest examples of a constraint structure. They will be used both to demonstrate a simple mathematical interpretation of the essential definition of architecture and to gain insight into its careful arrangement of wording.

For the case of a single real-valued variable, a constraint is then a type of relational property that defines a collection of intervals. There are four types that define intervals in one of the following four forms:

i. Closed intervals: [a, b]
ii. Open intervals: (a, b)
iii. Semi-open intervals
   a. Upper semi-open: [a, b)
   b. Lower semi-open: (a, b]

The corresponding classes of constraint structures will be denoted as IC, IO, IU, and IL. In engineering practice, the structure with open intervals represents strict constraints. The upper semi-open intervals are well-suited for the constrained operating space of the diesel engine in Figure 2.1. The inclusion of the lower bound in the intervals would correspond to the initial state that a cruise or acceleration starts from. In general, the semi-open intervals can be useful for constraints on time in dynamic systems. The inclusion of the lower bound would correspond to the start time of a process and the strict open upper constraint is a requirement to finish *before* a specified end time. Similarly, the inclusion of the upper bound in a lower semi-open constraint could signal that the process must finish exactly at the end time.

Only one architectural property will be needed to define a class of constraint architecture. This is jointness (expressed through set intersection). This is how feasible regions are formed in a design space. The mathematical statement of jointness is that the non-empty intersection of any two intervals in the specified structure is also a member of the structure. This means that every pair of constraints that are consistent (have non-empty intersection) can be replaced by a single constraint of the same form. In the above classification scheme of structures, only three of the four classes of structure satisfy this property. The class IC clearly does not. Consider two constraint intervals: [a, b] and [c, d]. The joint constraint is defined by intersection of the two intervals. Consider the case of $c \le b$; the intersection is non-empty and would result in the interval [c, b]. If $b=c$, the intersection degenerates into a point and is not an interval.

Thus, IO, IU, and IL are suitable for specifying joint constraints but IC is not. The architect then has three classes of structure to choose from for the purpose of specifying a *constraint architecture* for the design space and recommending it to the engineering team. The final choice can be made based on other criteria. For the constrained operating space of the diesel engine in Figure 2.1, the statement 'without violating the emissions constraints' in the textual descriptions associated with the joint constraints would correspond to the class IL. Violating a constraint $x=b$ in the operating space means $x>b$. Thus, the operating interval would be in the form $(a, b]$. However, based on the architectural analysis of the operating space, it might be more useful to choose the class IU so as to use the initial states more effectively.

3.3.2 CORRECT USE OF MATHEMATICAL LANGUAGE

The use of language in the application of architecture to the constrained operating space is correct and precise. However, as with any practice of engineering, many underlying details have been suppressed so as to make clear the message to the intended audience of the discussion. A skilled architect should be able to speak and write ‘bi-lingually’ in this way. In what follows are the details of the correct use of mathematical language that underly the constraint architecture example for the operating space of the diesel engine.

First, it is important to understand the relation of the mathematical terms class and type, and to grasp the distinction. Annex A-1 provides a distilled but comprehensive background on this and other essential mathematics that underly the concepts presented in this book. In general terms, a class is a collection of (mathematical) objects that is defined by a property. The class can be said to implement the property, but it is not the property. In the separation of the counting numbers into even and odds numbers, the property that defined the partition was ‘divisibility by the number two’. Even and odd numbers are classes of counting numbers that are realised (instantiated) in a set of counting numbers. In a constraint structure, the property is the constraint type. For a single real-valued variable when the pairs of distinct bounds (a, b) are varied over all values in which $a<b$, the result is a collection of intervals that are subsets of the set of all real numbers. The collection is a relational structure (as introduced in Chapter 2, Section 2.3).

When properties are used for classification schemes they are often referred to as classifiers. The term *type* is a property that is generally used in this sense. Thus, the even numbers are a type of counting number. This can be represented symbolically as «even», i.e., the name of the property that defines the set of even numbers as a subset of the counting numbers. This is referred to as a *type specification*. Constraint structures in the example belong to a class of structures defined by the type «unary relational». This level of formality is needed to formally explain that not all properties can be implemented in a given class of structures.

The associated type specifications can be denoted as «closed unary relational», «open unary relational», «upper unary relational», and «lower unary relational». In the language of the essential definition of architecture, these are four structural types that can be paired with one or more architectural property to define *constraint architecture*. When the property is consistent with the structural type it can be implemented in a class of structures defined by the type and becomes *a class of constraint architectures*. More simply it will be referred to as *a constraint architecture* when it has been realised in a set. In the example above for the constrained operating space of the diesel engine, only three of the four structural types were consistent with the architectural property of joint constraints.

As discussed in Chapter 2, Section 2.3, in the practice of science and engineering this formality is normally suppressed but if the models and concepts do not conform to this formal basis, they will be logically incorrect and exhibit errors. The reader is challenged to interpret the

ISO and OMG definitions of system architecture (provided in Sections 3.2.3 and 3.2.4) into this straightforward example or in any other mathematically meaningful way. On the other hand, the mathematical interpretation of the essential definition (provided in Section 3.2.5) is seen to be clear, precise, and indeed powerful because of the breadth of interpretations into real engineering problems that it can support.

3.4 LOGICAL MODELS OF CONCEPTS

The purpose of this part of the chapter is to show how natural language definitions of terminology can be modelled with the same level of formality as the science-based example in the Chapter 2. The ISO/IEC/IEEE 15288:2015 definition of system will be used as an example.

The system and architecture definition processes specified in Chapter 4 require precise yet intuitive definitions of the two terms system and architecture that have been normalised against each other. The logical model of system presented in this chapter will be evolved using the essential definitions. The result will be an integrated model of the concepts of system and architecture that is suitable for the specification of mathematically based technical processes and structured methods that implement the processes.

Concepts expressed in natural language can also be interpreted into sentences in the predicate calculus and then into a model. The process of doing so exposes the inconsistencies and ambiguities that occur in the common usage of natural language. This is an example of the Principle of Definition. This type of model, like any other model, may not be unique but it serves as a point of reference for analysis and discourse that should lead to an agreed upon model. The correct use of a formal language in this way is beyond the grasp of all but perhaps a few experts.

Early progress towards making such a process of expression more intuitive was made by Peter Chen (1976) in the 1970s using a graphical language, Entity–Relationship (E–R) Diagrams. These are still in use today for relational data bases. Around this same time period, John Sowa (1983) developed a similar type of diagram that was specialised to the predicate calculus. These were called Conceptual Graphs, which he used to develop a framework of Conceptual Structures. As presented by Sowa, this is essentially a process for representing knowledge inside computer systems; but it is more general and can be applied more widely. In his framework, a concept is a mental interpretation of a percept (an impression of the physical world or a thought about an abstraction). Basically, any thought about something (a referent) that is capable of being expressed in natural language can be regarded as a concept in this more formal sense. Conceptual Graphs have been standardised as part of information technology for many years. The most recent version is ISO/IEC 24707:2018 (ISO 2018).

By the late 1990s, the notations of E–R Diagrams and Conceptual Graphs had been absorbed into the UML that has been a standard from the Object Management Group (OMG) for object-oriented software development for over two decades. It can also be used to represent sentences in the predicate calculus in a way that is intuitive and graphical. Entities are represented as classes, which in the graphical notation of UML are rectangles with a name (typically a noun) that identifies the class. A class is a collection of objects of the same type. These can correspond to the variables in a formal predicate. The relation between the variables, which in mathematical logic is represented by a predicate letter, is resolved into relationships between the variables which are represented by lines between the classes and typically annotated with a verb or verb phrase that expresses the meaning (semantics) of the relation.

A concept expressed in natural language that has been interpreted in terms of a sentence in the predicate calculus will be referred to as a *logical model of the sentence*. The objective of the logical modelling of a sentence is to extract the relations comprising the sentence using a formal language to derive a minimal model that is complete and captures the intended meaning of the sentence. Key words are selected that remain undefined in order to avoid the circularities possible in natural language. This approach follows the axiomatic approach of mathematics. See the foundational work (Dickerson 2008) that linked the scientific concept of a mathematical model in Hamiltonian mechanics with the concept of model in mathematical logic using a procedure for the logical modelling of sentences. This procedure was demonstrated in detail in the first edition of this book (Dickerson and Mavris 2010) and is provided in Annex A-2.

The outcome of the procedure will now be illustrated using the term system. The definition currently adopted in the most widely accepted standard for systems is (ISO 2015) "a combination of interacting elements organized to achieve one or more stated purposes". Although this particular standard has not yet adopted the use of logical models of sentences, numerous others have, e.g., ISO/IEC/IEEE 42010:2011 (ISO 2011) to name just one. The diagram depicted in Figure 3.1 of this particular definition is organised around six key words, only one of which is a verb (achieves). Two of the adjectives have been converted to nouns and are represented as properties possessed by two of the nouns occurring in the defining sentence. The third adjective (stated) has been suppressed and would be best represented as a verb (state) but there is no suitable noun in the sentence to serve as the subject, e.g., 'Stakeholders state'. Altogether, six relationships form a predicate that has been expressed by the natural language sentence. As a formal sentence in logic, the overarching predicate statement would be:

$v=P(u1,u2,u3,u4,u5)$ where v takes on the value System, and $u1,u2,u3,u4,u5$ take on the values Purpose, Combination, Element, Interaction, and Organisation, respectively. In this formal sentence the equal sign ($=$) is interpreted as 'is'. The statement is a well-formed formula in which every variable has been assigned a value, i.e., the formula has been quantified. It therefore is a sentence in the predicate calculus.

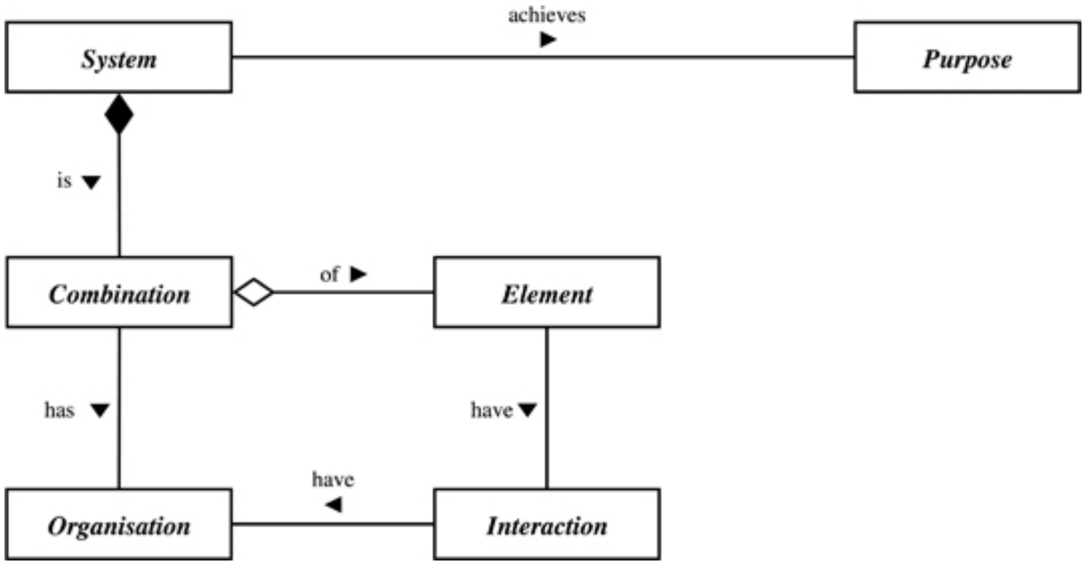

FIGURE 3.1 Logical model of a standards-based definition of system

The predicate letter P can be resolved into five further predicates of two variables each. For example, $P1\ (v,\ u1)$ could be the predicate 'System achieves Purpose'. The resulting formula would be

$$P1\ v,\ u1 \wedge v=u2 \wedge P2\ u2,u3 \wedge P3\ u3,u4 \wedge P4\ u2,u5 \wedge P5\ u4,u5.$$

In this predicate formula, the symbol ∧ is the 'logical and'. Thus, all six relationships in the diagram are represented in the predicate formula in terms of a conjunction ('logical and' statements). In practice, this level of technical detail would be suppressed. However, the point is that the graph in Figure 3.1 is a structure in which a sentence in the predicate calculus has been interpreted. The UML diagram in the figure therefore represents a first-order model in the mathematical sense and has the precision and consistency needed for mathematical analysis. The UML notation in the diagram also has symbols for aggregation and composites that can be useful for modelling systems. These will be explained in Chapter 8.

3.5 MODEL-BASED SYSTEMS ENGINEERING

Now that an accepted definition of system has been logically modelled and a definition of engineering has been proposed, it would be appropriate to reflect upon an appropriate definition of the term systems engineering. One approach would be to use the technique of viewpoints and views that are commonly practiced in engineering. The result from doing this will be compared with an accepted statement of model-based systems engineering. Recall that a viewpoint is a technique for abstraction to focus on particular concerns regarding a concept. The term abstraction as used here means the process of suppressing selected detail to establish a simplified model. A viewpoint can be thought of as a perspective. A view, on the other hand, is a viewpoint model, a representation of the concept from the perspective of a particular viewpoint.

As noted in the first edition of this book (Dickerson and Mavris 2010), it is very natural to then consider that

Systems engineering is the practice of engineering from a systems viewpoint.

This definition could be used with any definition of engineering. It also has a syntax that is consistent with the terminology of other engineering disciplines, for example:

- Chemical engineering could be considered as the practice of engineering from the viewpoint of chemistry.
- Structural engineering could be considered as the practice of engineering from a structural viewpoint.

This syntax also makes clear the difference in roles between systems engineers and other engineers in the multi-disciplinary environment in industry.

As demonstrated in the logical model in Figure 3.1, the term system can be expressed as a well-defined concept. Therefore, it would also be natural to replace the term concept in definition of engineering presented earlier in this chapter with the term system:

Systems engineering is a practice of system realisation in which relations between structure and functionality are modelled using the laws of science for the purpose of solving a problem or exploiting an opportunity.

This definition is also consistent with the concept of *engineering of systems*. All three of these concepts are model-based by the nature of the definition of the term engineering given earlier in this chapter.

The 2007 INCOSE Vision 2020 as noted in the INCOSE Systems Engineering Handbook defined model-based systems engineering as, “the formalised 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” (INCOSE 2015). The scientific basis of engineering expressed in this chapter together with the formal first-order model theory that has been demonstrated for the concepts of system in this chapter clearly establishes the approach taken in this book as a qualified approach to model-based systems engineering, especially when these concepts are used to implement the technical processes of accepted standards such ISO/IEC/IEEE 15288:2015 (ISO 2015). This implementation will begin at a foundational level in the next chapter and be carried forward with increasing detail throughout the book.

4 Structured Methods

DOI: 10.1201/9781003213635-4

KEY CONCEPTS

Separation of concerns
Association and cohesion
Model transformation
Synthesis of models

The basic concepts of Structured Analysis and Design were reviewed in the brief histories of architecture and model-based systems engineering in Chapter 1. Initially introduced by Yourdon, these concepts have evolved and become the basis of standards and practices in the twenty-first century such as the OMG Model Driven Architecture for software development. The *principle of separation of concerns* is fundamental to these concepts and is broadly applicable to engineering and systems. Architecture, analysis, and transformations between models are central to a structured approach to system design. The logical foundation of rigorous methods for model specification and transformation used throughout this book is based on a synthesis of the concepts introduced in Chapter 2 with those of Yourdon. This is the basis of a relational framework for Structured Analysis and Design.

A system realisation of the concept for an emissions control governor (ECG) investigated in Chapter 2 will be used in this chapter to illustrate how the specification and transformation of first-order models can be used for precise interpretation of a science-based solution into a model-based specification of an information-intensive systems solution. This will complement and illustrate the foundational discourse of how systems, architecture, modelling, and engineering can be bound together in a straight-forward intuitive way that is rigorous and practical. The remainder of the book will be concerned with development and explanation of the requisite technical details for application to more complicated problems than the simple examples in this foundational section.

4.1 SCOPE OF CONCERNS AND SYNTHESIS OF CONCEPTS

Two core technical processes specified in ISO/IEC/IEEE 15288:2015 (ISO 2015) are explored in depth in this book: System Requirements Definition and Architecture Definition. Altogether there are three requirements technical processes in the ISO standard. The other two are Business or Mission Analysis, and Stakeholder Needs and Requirements. However, an in-depth exploration of all three is beyond the scope of the book and indeed could constitute a separate book. In the tutorial and case study materials, the outputs from the other two processes for requirements are assumed (given as starting points). Core elements of the Unified Modeling Language (UML) that are shared with the Systems Modeling Language (SysML) will be used for specifying essential system models and architecture. Limiting the initial specification of models to UML serves two important purposes. First, as seen in Chapter 2, is that its form is close to that of mathematical logic and set theory. Simple guidelines and procedures can therefore be given that make system models and architecture more rigorous and traceable because they reflect an underlying expression in the predicate calculus. The other purpose is that the core models specified in UML can be given directly to software engineers for code development. The same models can be extended or elaborated into SysML for use by other engineers for design and development. If maintained properly, these 'essential models' can be used to maintain the conceptual integrity of the system through a comprehensive architecture that enables the design and development of hardware and software in a holistic concordant way.

4.1.1 ALIGNMENT OF DEFINITION PROCESSES TO SYSTEM LIFE CYCLE

The 14 technical processes specified in ISO/IEC/IEEE 15288:2015 are easily aligned to the traditional Systems Engineering Vee as in Figure 4.1. This arrangement lends itself to a 'top-down' approach that progresses from the top left of the Vee then down the left side using the traditional method of system definition and decomposition. Implementation of the system elements of the design is at the bottom of the Vee. Integration and Verification then progress in an iterative loop until prototypes and the finished system are ready to be validated. After validation is accomplished, the system is transitioned to the customer where the life cycle is completed (deployment through disposal).

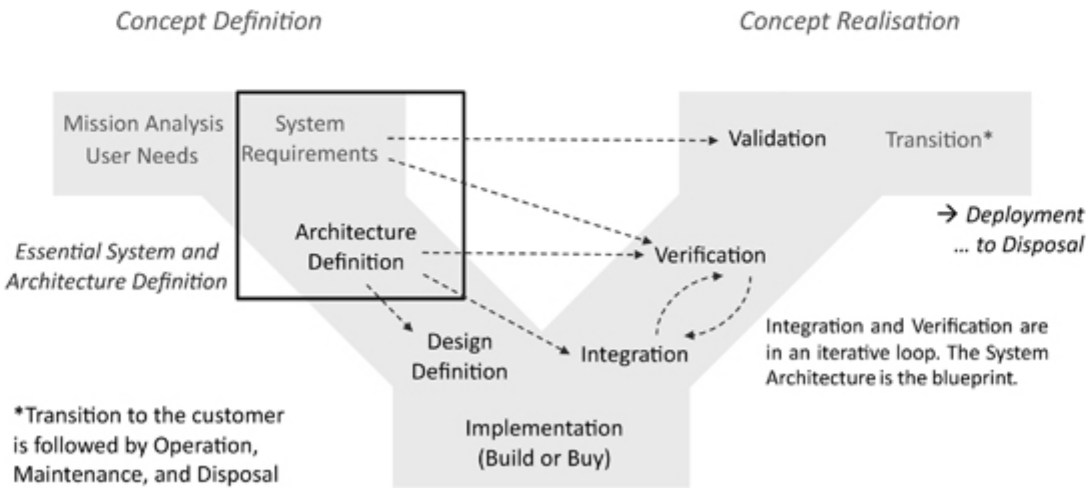

FIGURE 4.1 Alignment of Essential Definition processes

This process was originally developed in the mid to late 1990s in the context of large-scale systems in the defence and aerospace industry that had long development cycles. Thus, the linear progression through the Vee was not a significant factor in system development. However, as systems engineering expanded from these origins into the commercial domain, this has become an issue. This is especially true for software development. In the model-based approach to architecture and systems engineering presented in this book, the process of system definition and decomposition will be subsumed into one of model specification and transformation. This permits both vertical and horizontal reach from Architecture Definition throughout the Vee, not just to Design but also to Integration, Verification, and Validation. The outcome of the definition process should be a model-based specification of the system that is ready for design and might also be ready for implementation (i.e., sufficient detail for 'build or buy' of the system elements). Furthermore, in this approach a linear progression through the Vee need not be followed; thus avoiding many of the issues associated with applying it to complex systems and commercial development.

How this can be done in a commercial environment was a subject of a five-year research programme with an automotive original equipment manufacturer (OEM) that was jointly sponsored by the government and the OEM (Dickerson and Ji 2018). The research was concerned with virtual design and prototyping for increased speed to market. The long-standing practice of specifying configuration items (CIs) during Architecture Definition can also be employed to an advantage in the model-based approach. CIs are hardware, software, or composite elements at any level of the system hierarchy designated for formal control of form, fit, or function. They are characterised by being replaceable as an entity, having defined functionality, and a unique specification (INCOSE 2015). CIs can and should be specified as model elements in the system model. This provides a criterion for a minimal level of modelling because every CI is managed under formal control.

Defining a system as either a combination of interacting elements or simply a set of elements initiates a *System and Requirements Definition* process that begins with a simultaneous decomposition process that identifies the relevant elements. The definition process must then use this identification to transform the outcomes of the mission analysis and stakeholder needs processes into a technical view of the system. This includes

- System boundary, elements, functionality, and process specifications
- Functional, performance, and non-functional requirements; and constraints
- Traceability of system requirements to stakeholder requirements

The specifications, requirements, and constraints are system concepts.

Architecture Definition is concerned with how stakeholder and system requirements relate to system structure. The salient features are:

- Defining structures associated with a system to implement the system concepts
- Stating the concepts as properties to be implemented in the structures
- Interpreting the statements into the structures (i.e., specifying models)
- Defining relations amongst the structures to include transformation and synthesis
- Normalisation of concepts, semantics, and relations across the models
- The structural relations include interfaces between the processes of the system elements

A process of model specification and transformation then follows from the interpretation of the concepts into the structures and the relations amongst the structures.

In order to use an architecture defined by this process to enable Design Definition and implementation specification, it must be further refined to define how:

- Structures associated with the system response under intended and alternative conditions
- Control structures can be defined for precise implementation of responses

The outcome of the above definition and refinement processes is a collection of models that are bound by a system architecture that is robust, and are suitable for specifying implementation of the hardware, software, and composite elements of the system, or to enable an iterative design definition process that does. This collection of models bound by the architecture will therefore be referred to as an Implementation Model. A holistic comprehensive system architecture will include a software architecture that can be deployed to software developers. Deployment to state machines will define how structures associated with the system change state in response to the occurrences of events.

4.1.2 MODELLING LANGUAGES AND LOGIC

The organisation of diagram types in SysML and their foundation in UML is depicted in Figure 4.2. SysML is a general-purpose modelling language similar to the UML that is formally defined as a profile of UML 2.0. This means that the SysML metamodel (OMG 2017a), i.e., the syntax, semantics and modelling rules are built around the metamodel of UML (OMG 2017b) by means of extensions. These are in the form of stereotypes, tagged values and constraints. Specifically, SysML inherits a subset of the model elements defined in UML and extends other model elements to facilitate generic system modelling. The commonalities and differences between the two modelling languages can be better visualised at the level of diagram types rather than at the level of model elements. Adapted from SysML specification, Figure 4.2 provides this visualisation from the perspective of SysML diagrams. Note that the SysML Block Definition diagram and Internal Block diagram are analogous to the UML Class diagram and Composite Structure diagram (OMG 2017b) respectively with moderate modifications. Other well-known UML Profiles include, for example, the UML profile for the Modeling and Analysis of Real-time and Embedded Systems (often abbreviated by MARTE) (OMG 2019). This is a modelling language that extends UML specifically for application to embedded systems.

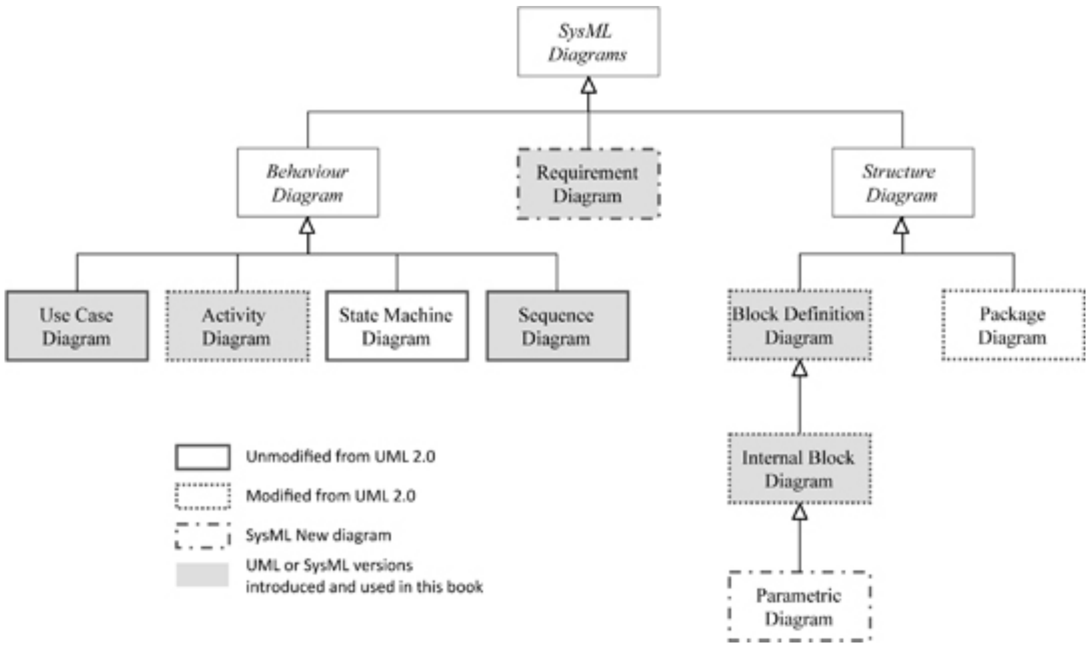

FIGURE 4.2 The Systems Modeling Language

The essential definitions of structure, architecture, and system presented in Chapter 3 have a mathematical basis that is suitable for deriving technical processes for systems engineering (Dickerson et al. 2021). The underlying concepts of structure and behaviour are more general than those of the SysML diagram types of the same name. As noted earlier, the core diagrams of UML that are in common with SysML can be used to specify system models and architecture that are mathematically based. This is similar to how a definition of 'system' was modelled with UML Abstract Classes in Chapter 3, yielding a graphical representation of a logical predicate formula. Use Case diagrams are particularly straight forward to model with this level of rigor when some simple guidelines and procedures are adhered to. Because this type of diagram is the simplest and most ubiquitous in systems, software, and standards, it merits a detailed explanation.

A Use Case diagram only has four types of graphical elements: a figure (which represents an UML Actor), a line (which represents an UML Association), a rectangle (which depicts the system boundary), and an oval (which represents an UML Use Case). It is a widely accepted practice that each use case should be written as a verb-noun phrase. Recall from Chapter 3 that this type of phrase is a predicate. Thus, similar to the logical modelling of sentences, use cases can be written as predicates $P(v)$ that associate elements of the environment (actors) with the system as a black box entity. The association can be expressed as a sentence in one of two general forms: (i) System <interacts with Actor>, or (ii) Actor <interacts with System>. The logical formula for the sentence is $u \wedge P(v)$ where the predicate variables u, v each exclusively take on the value of either the actor or system name. (The symbol $\wedge$ is the 'logical and'.) The formula is indeed a sentence in the predicate calculus because the variables have been quantified (by the actor and system names). All of these concepts and notations are captured succinctly in Figure 4.3.

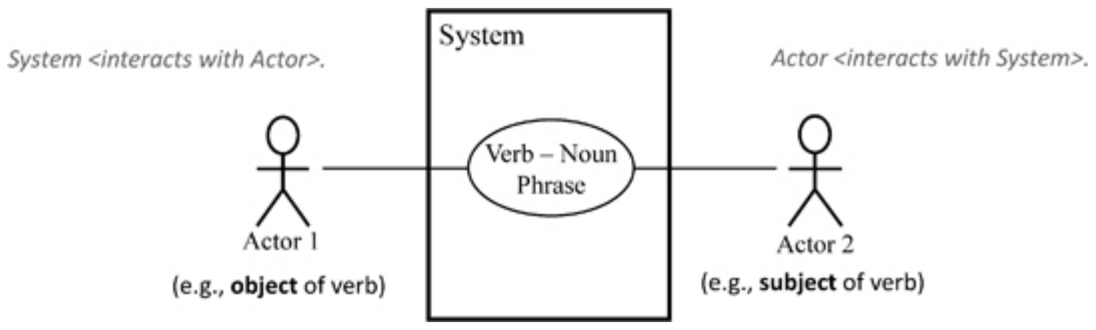
System <interacts with Actor>.
System
Actor <interacts with System>.
Verb – Noun Phrase
Actor 1
(e.g., **object** of verb)
Actor 2
(e.g., **subject** of verb)

FIGURE 4.3 Relation of predicates to use cases

4.1.3 FUNCTIONAL REQUIREMENTS FOR THE ADAS ECG

As previously noted, the Use Case diagram is the most widely used diagram in UML and SysML. Functional requirements can be easily and precisely captured in this simple graphic when written as simple sentences using predicates. Any *system* realisation of the concept for the ECG investigated in Chapter 2 must include a functional requirements specification. The annotated diagram in Figure 4.4 fulfils this need. Because it follows the guidelines for the relation of predicates to use cases, it offers textual (prose) statements that can be represented formally in the predicate calculus but are presented in a simple intuitive graphical form that is easy to communicate to customers and engineers alike. This is precisely the intent of the Principle of Definition and gives the system architect a means to fulfil their role regarding conceptual integrity of the system; thus, adhering to two of the principles of architecture and systems engineering.

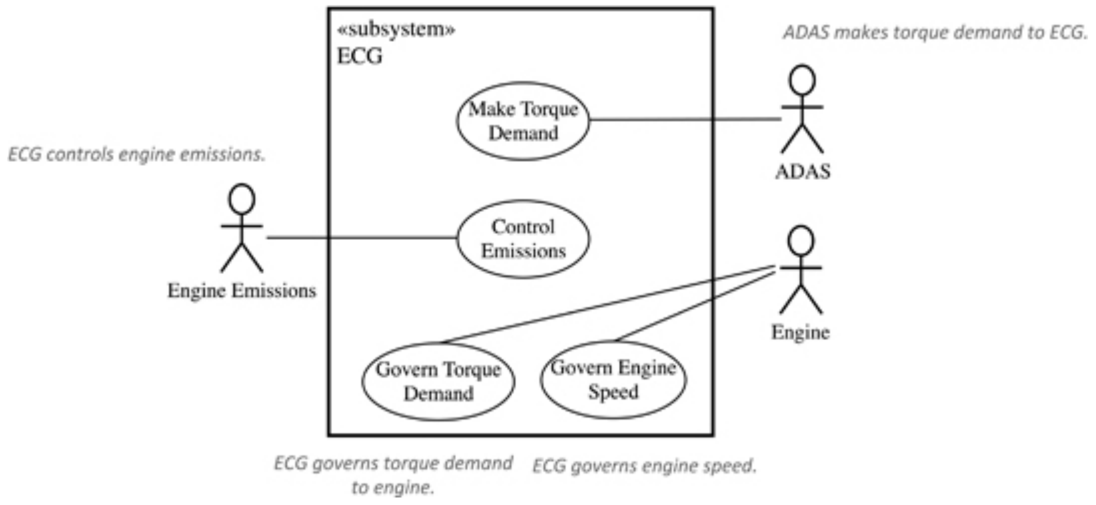
«subsystem»
ECG
ADAS makes torque demand to ECG.
Make Torque Demand
ADAS
ECG controls engine emissions.
Control Emissions
Engine Emissions
Engine
Govern Torque Demand
Govern Engine Speed
ECG governs torque demand to engine.
ECG governs engine speed.

FIGURE 4.4 ADAS ECG Use Case diagram

Complementary to the Use Case diagram in Figure 4.4 is the matrix representation in Table 4.1 that depicts the information content of the diagram. The specific interpretation of the symbols is as follows:

- Actors:
  - A1=ADAS
  - A2=Engine Emission
  -

    - A3=Engine
- Predicates:
    - P1v1=Make Torque Demand
    - P2v2=Control Emissions
    - P3v3=Govern Torque Demand
    - P4v4=Govern Engine Speed
- Sentences: W(Ai,S;u∧Pv)
    - Ai=ith Actor
    - S =EGC System (black box)
- Logical Symbols: ∧, ∨, ¬
- Quantification: u, v exclusively take on the value of Ai or S

TABLE 4.1
ADAS ECG Use Case Diagram in Matrix Form

Actors	Predicates	P1(v1)	P2(v2)	P3(v3)	P4(v4)
A1		W1,1	…	…	…
A2		…	W2,2	…	…
A3		…	…	W3,3	W3,4

Written in symbolic notation this representation can be machine readable. This sort of representation should be running in the background of any commercial modelling tool for a practice of engineering that uses UML and SysML.

When the traditional process of system definition and decomposition is subsumed by one of model specification and transformation, graphs and tables such as in Figure 4.4 and Table 4.1 can be interpreted as matrices that can be used for various types of analysis. For initial modelling of use cases the matrices are very simple. The matrix for the actors is an $m\times 1$ vector (in the table, $m=3$) and the matrix for the use case predicates is a $1\times n$ vector (in the table, $n=4$). The $m\times n$ matrix depicts how the actors are associated with the use case predicates by the sentences. Because the sentences establish interrelationships between elements of the environment and properties of the system (e.g., functionalities expressed through use cases), the model represented by a Use Case diagram is an elementary one that expresses the concept of system functionality as a set coupled with an environment via interrelationship.

In general, there can also be relations between the actors, as well as relations between the predicate variables. In this case, the one-dimensional vectors for the actors and predicates must be expanded into $m\times m$ and $n\times n$ matrices, respectively. This is depicted by the relational framework of matrices in Figure 4.5, which is formed by three matrices, M, Q, and N. Rather than listing the actors and predicates, as in Table 4.1, the variables of the predicates and variables associated with the actors are listed. In the design of systems, the variables are generally metric variables that quantify the requirements and the system level characteristics.

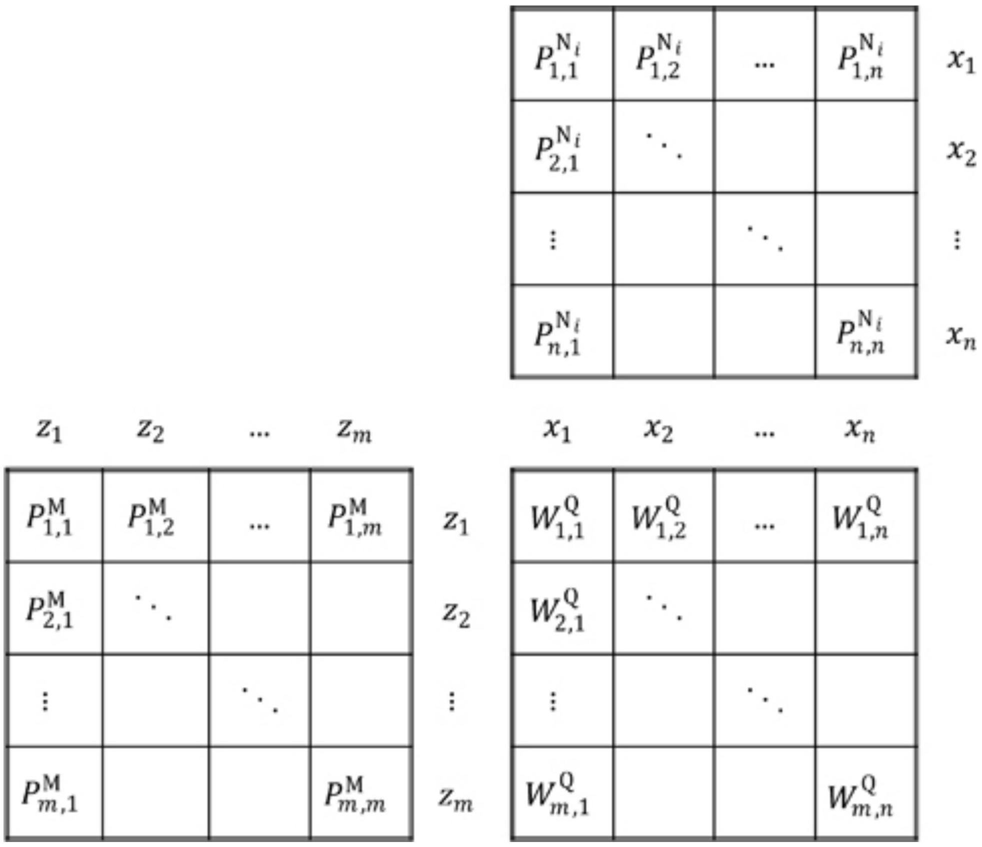

FIGURE 4.5 Relational framework

For the specification of system models, the M-matrix is an array of relationships between design objectives z1, z2, … zm and the N-matrix is an array relationships between the system level characteristics x1, x2, … xn. The relationships in the cells of the matrices are the predicates. As with the use case table, the predicates in the two matrices are joined by sentences. These are in the m×n Q-matrix. The complexity of system design and specification now becomes evident. Furthermore, the interrelational nature of systems adds a further complexity: the sentences in each row of the Q-matrix can convey relationships between the objectives into relationships between the system characteristics. These will be in conjunction with any preconceived relationships between the characteristics. Consequently, the interrelational nature of systems implies that there is a family of N-matrices that must be used for analysis and design of the system (i.e., determining the best values of the system characteristics to achieve the design objectives). This is why the superscript of the predicates in the N-matrix in Figure 4.5 is indexed by the rows of the M-matrix. This family of matrices {Ni} is in fact a *relational structure* on the system design characteristics {x1, x2, … xn}. Thus, the first-order model theory introduced in Chapter 2 is not just an interesting insight. It is at the very core of architecture, analysis, and system design. Furthermore, the Q-matrix becomes a model transformation from the logical structure of the problem to be solved into the relational structure of the system.

4.2 STRUCTURED ANALYSIS

Chapter 1 introduced the Yourdon Structured Analysis method which is based on a concept of separation of concerns. The Principle of Structured Analysis can be summarised succinctly as follows:

The specification of the problem should be separated from that of the solution.

Two types of system models are used in structured analysis:

Essential Model (implementation-free representation)
Implementation Model

These two types of models are linked by interpretation of a behavioural model into one for implementation that is a realisation of the essential model. The Implementation Model is a specific representation of the hardware and software needed to realise the system.

The Essential Model is comprised of the Environmental Model and the Behavioural Model (Yourdon 1989). The Environmental Model defines the boundary between the system and the environment. It provides a short description of the system purpose, a context diagram, and an event list. The Behavioural Model identifies what the flows are through the system and establishes controls over the system flows. In this way, a model of what is flowing through the system is developed. It also determines the possible system states and transitions.

4.2.1 FRAMEWORK FOR STRUCTURED ANALYSIS

Structured Analysis can be implemented through a framework of model specification and transformation as depicted in Figure 4.6. This corresponds to the technical processes in the upper left corner of the Vee diagram in Figure 4.1 and proceeds in two steps. Based on the essential definitions, the system is formed of two sets of elements: the system elements and the elements of the environment. The System Requirements Definition process from ISO/IEC/IEEE 15288:2015 (ISO 2015) can be applied to transform a specification of the environment based on mission analyses and user needs into a technical view of system functionality. If the resulting technical view of the system has sufficient detail to define the functionality needed for all users of the system, then this step in the structured analysis process is finished (subject of course to agreement between the stakeholders). Otherwise, further elaboration of functionality is needed. This can entail identifying and defining new functionalities not previously discovered in the earlier Requirements Definition processes or applying system decomposition to specify lower level functionalities. The output of this first step in the structured analysis process is the Environmental Model. Figure 4.4 provides an example for the ADAS ECG.

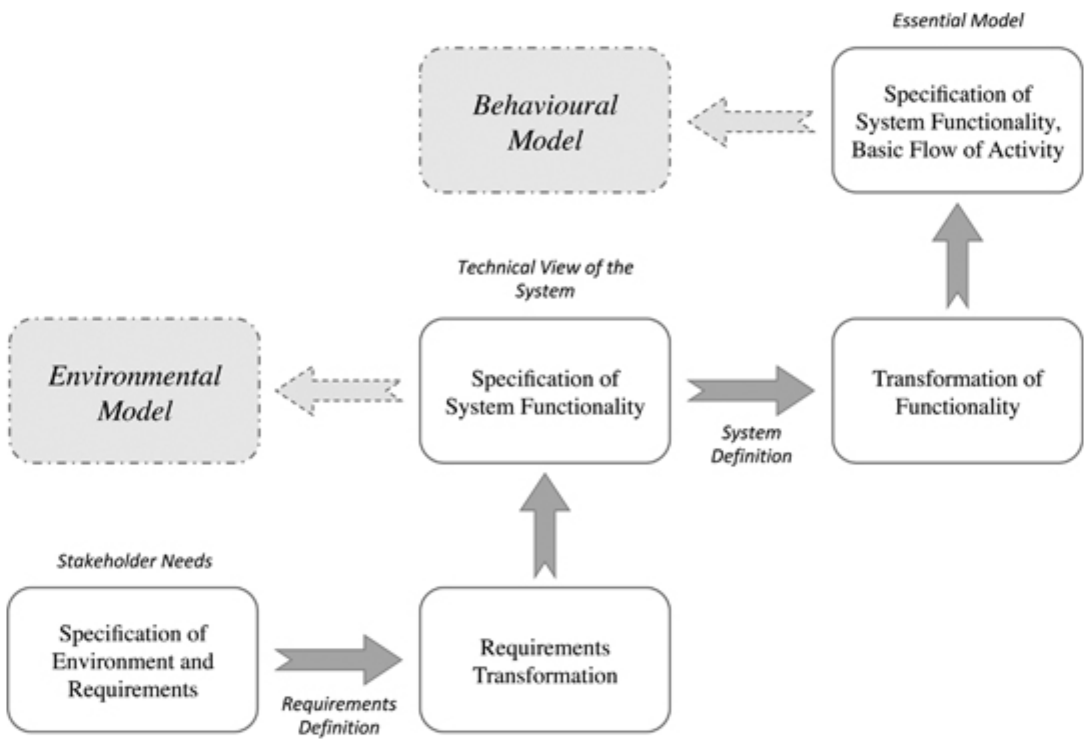
Essential Model
Behavioural Model
Specification of System Functionality, Basic Flow of Activity
Technical View of the System
Environmental Model
Specification of System Functionality
System Definition
Transformation of Functionality
Stakeholder Needs
Specification of Environment and Requirements
Requirements Definition
Requirements Transformation

FIGURE 4.6 Framework for Structured Analysis

The second step, as indicated in Figure 4.6, is to transform the system functionality into system behaviour (which shows the internal processes). This process begins when system functionality has been defined in sufficient detail to enable specification of system activities and basic flow of actions that enable each functionality. When the basic flows have been specified to a sufficient level of detail that exhibits how the system processes will deliver the intended functionality in the intended environment, the behaviour modelling is complete. The fourth part of this chapter provides an example of basic flow of actions for the ADAS ECG. The specification of the Behavioural Model completes the specification of the Essential Model. The structured analysis process should be applied to the system elements in the same way as it is to the (overall) system.

4.2.2 MODELS FOR STRUCTURED ANALYSIS

The Requirements and System Definition processes can be implemented by model specifications and transformation to create the Environmental and Behavioural Models. For the Environmental Model, the Use Case diagram provides a graphical model that defines the boundary between system and environment, and provides a description of the system functionality (purpose). The transformations are: Use Cases # Included Use Cases (functional decomposition) # Use Case descriptions. For the Behavioural Model, the Activity diagram provides a graphical model that identifies what the flows are through the system and will establish controls over the system flows. Thus, two classes of structure are realised in the Essential Model: a partition (of the actors and system in the use cases) and a basic flow structure (in the Activity diagram).

4.3 STRUCTURED DESIGN

The model-based approach for software development from a behavioural viewpoint introduced by Yourdon (1989) is an approach to implementing the Essential Model of the system:

Systems should be comprised of modules each of which is highly cohesive but collectively are loosely coupled.

This succinctly states the Principle of Structured Design. Cohesion minimises interactions between elements not in the same module, thus minimising the number of connections and amount of coupling between modules. The strongest form of cohesion in the Yourdon approach was based on functionality. His ranking was only suggestive and not based on any sort of rigorous analysis. The following is a slight re-ordering of Yourdon list with some explanation:

Coincidental	*concurrence without apparent cause*
Logical	*e.g., association by conjunction (*p *and* q)
Sequential	*association by order (e.g., 1 precedes 2)*
Temporal	*association by time order*
Communication	*exchange of information*
Procedural	*related by a specified 'way of doing'*
Functional	*bound by a 'way of doing' for purpose*

Coincidental association is the weakest type of cohesion. Logical association is the first meaningful type. Functional binding is the strongest form of cohesion in Yourdon's ranking.

4.3.1 FRAMEWORK FOR STRUCTURED DESIGN

Structured design can be implemented through a framework of model specifications, transformation, and synthesis as depicted in Figure 4.7. Architecture Definition links the technical processes in the upper left corner of Figure 4.1 to Design Definition and proceeds in two steps. Based on the essential definitions, interrelations between the two sets of elements that form the system (elements of the system set and elements of the environment) need to be specified. In the structured analysis process, the behaviours of the system were defined using relations between the system and elements of the environment. These relations were modelled with use cases that were elaborated into functionalities that were actionable. The first step of structured design is to modularise the actions and make them executable.

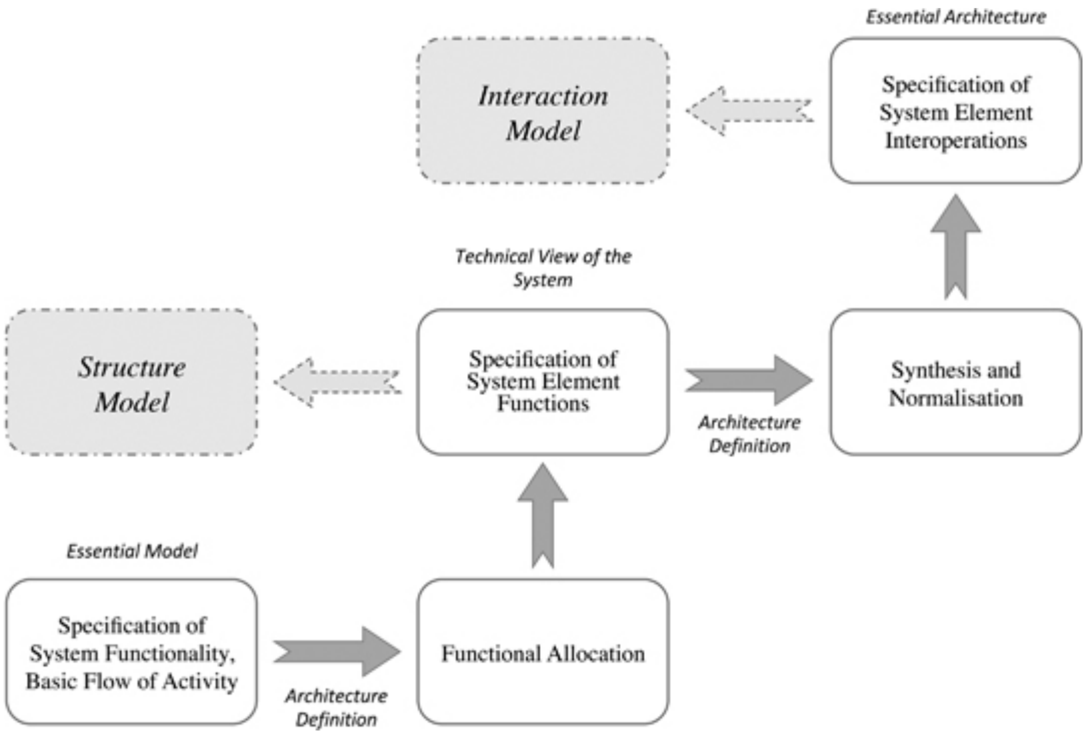
Essential Architecture
Interaction Model
Specification of System Element Interoperations
Technical View of the System
Structure Model
Specification of System Element Functions
Architecture Definition
Synthesis and Normalisation
Essential Model
Specification of System Functionality, Basic Flow of Activity
Architecture Definition
Functional Allocation

FIGURE 4.7 Framework for Structured Design

Various properties can be attributed to the system elements, but it is the functions of the elements that implement the intended actions. These are also called operations, which is the term used in the specification of a UML Class. This type of relation between the actions of the system and operations of the system elements is commonly referred to as *functional allocation*, which is depicted in the lower left Architecture Definition transformation in Figure 4.7. The distinction between functionality and functions (element operations) is that functions are executable as input–output relations. Thus, the term *system element functions* in the figure reflects that system functionality can be implemented in the first-level hierarchy of the system decomposition. If this is not possible then the term should be replaced by *system element functionality*, in which case at least one further level of hierarchy will need to be defined.

In a UML model of the system, the functional allocation will result in a class structure that has implied and intended exchanges between the operations: every input to an element operation must be supplied by another element (either from the system set or the environment), and every output should be consumed by another element. Modularisation of the system elements is accomplished by the UML Classes, which can be implemented in either hardware or software. Class definition of the elements is not specific as to the details of the realisation (e.g., the traditional concept of 'build or buy' at the bottom of the Vee in Figure 4.1). It is not necessary to use UML Classes to specify the elements but as seen in Chapter 3, the graphical notation provides a convenient intuitive method to represent the underlying mathematical classes and logic of a conceptual model of the system.

Before proceeding to the Essential Architecture in the figure, all the previous model specifications must be normalised, i.e., there must be an equivalence of the semantics, syntax, and relations across the concepts and elements of the models that have been specified. This is similar to the traceability in a system and Requirements Definition process. The resulting structure of the models will then reflect not just how the system elements operate and are interrelated but also how they interoperate to achieve the intended purpose of the system. Figure 4.9 in Section 4.4 of this chapter provides an example of an interaction model for the ADAS ECG.

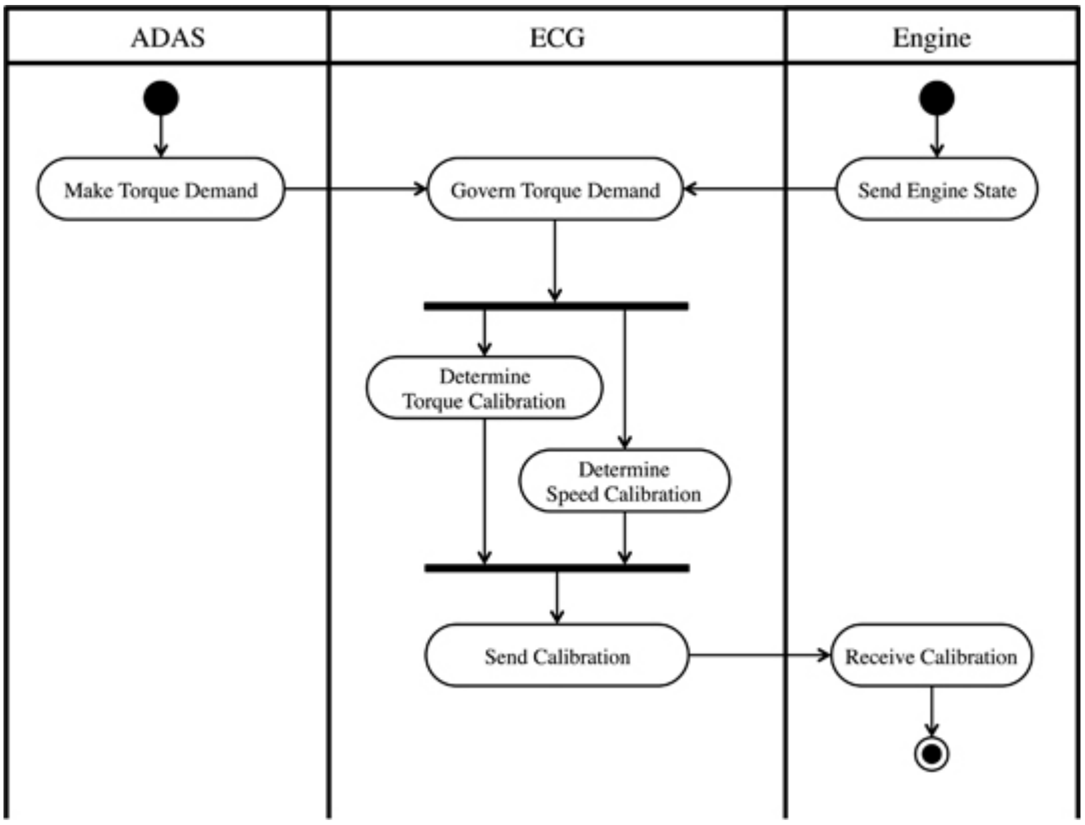

FIGURE 4.8 Essential system model of ADAS ECG

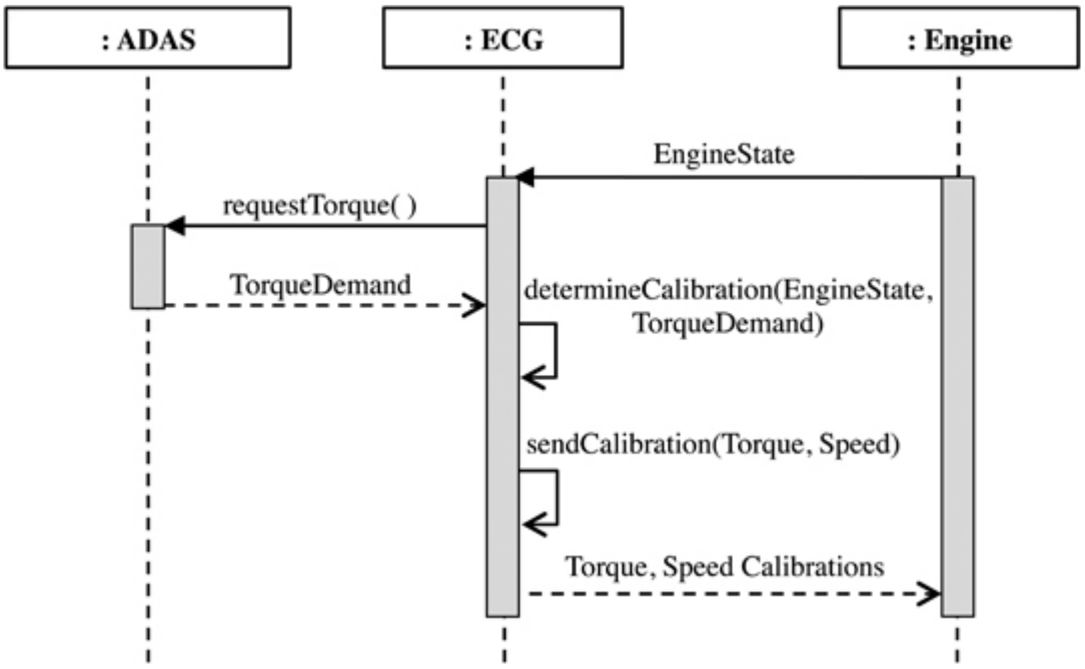
: ADAS
: ECG
: Engine
EngineState
requestTorque( )
TorqueDemand
determineCalibration(EngineState, TorqueDemand)
sendCalibration(Torque, Speed)
Torque, Speed Calibrations

FIGURE 4.9 Interaction model for the essential system architecture of ADAS ECG

4.3.2 MODELS FOR STRUCTURED DESIGN

A structured design process can be implemented by model specifications, transformations, and synthesis of the model elements of the system into an interrelational structure for the essential architecture. UML can facilitate the specification of two models that comprise the interrelational structure: the (Class) structure model and the (Object) interaction model. Activity diagrams specified in the essential model of the system are transformed into Class diagrams through a process of functional allocation that associates system actions with system element operations. The interaction model must account for all the required exchanges based on the input–output relations of the operations of the elements. The model is represented in UML (and SysML) by a Sequence diagram that synthesises the system elements (represented by UML Objects), sequencing of operations owned by elements, the interactions (e.g., exchanges) between elements, and the control structure that governs the flow. In terms of UML model elements, the model transformations are realised by the following associations: Class # Objects # Lifelines; Operations # Function Calls. The resulting interrelational structure for the *Essential Architecture* can be refined to specify an *Implementation Model* that enables Design Definition and system implementation.

4.4 REALISING THE SOLUTION CONCEPT FOR THE ADAS ECG

The frameworks for relational specification of models and their transformation, structured analysis, and structured design will now be employed to illustrate how to specify a system that realises the concept for an ECG investigated in Chapter 2. The specification and transformation of first-order models in the relational framework can be used for precise interpretation of the science-based solution into a model-based specification of an information-intensive systems solution. The Use Case diagram in Figure 4.4 serves as a model-based specification of functional requirements. The quantification of these requirements is supported by the response surface analysis organised around Figure 2.1 in Chapter 2. Two objective variables (CO_2 and CO) were analysed in relation to two system characteristics (engine states). A control strategy for how to control the states (engine speed and torque) was developed. In the system solution this becomes a control algorithm that mediates demands from the ADAS to the engine. How this relation is mediated by the ECG is made clear through the intended functionality expressed in the use cases. The development of the algorithm is the concern of a domain engineer. The system structure and requisite properties within which it is implemented is the concern of the architect. This includes the interface requirements for the ECG.

The inputs to and outputs from the ECG are also in the remit of the architect. In addition to functionality and its quantification (based on the use cases, regulatory requirements, and analysis of the constrained engine operating space), the architect must specify the flow of events between the ECG and the ADAS and the engine. For the purpose of this chapter, the synthesis of the essential model of the system represented by an Activity diagram into a Sequence diagram will be sufficient for illustrating the ECG specification as an information-intensive system.

The Activity diagram in Figure 4.8 depicts a design-level model for the system actions of the Engine, ECG, and ADAS that must be performed and the flow of events in their performance. Note that flow does not necessarily imply interaction between system elements. Analyses and design decisions play into the transformation of use cases into a functional flow (flow of actions). The details and demonstration of how this can be done are a subject of Part II tutorials. For this illustrative example, it is sufficient to note the similarities and differences with Figure 4.4. First, the overarching functionality is not explicitly represented in the functional flow. Rather, it is the intended outcome of the flow. Next, the functionality for the ADAS to make a torque demand to the ECG has not changed but the functionality of the ECG to govern torque demand and engine speed has been altered by design decisions. Specifically, the governance of torque demand has been resolved into two calibration actions. These are indicated by the actions between the horizontal bars in the diagram that are symbols for the forking and joining of actions. The actions between these two bars represent an algorithm to be developed by a domain engineer, the output of which is a pair of calibrations that are sent to the engine. Finally, the action to send the engine state to the ECG is not a functionality of the ECG. Rather, it is the outcome of an interface definition. The engine state is needed in order to perform the internal functions of determining calibrations.

The Sequence diagram in Figure 4.9 reflects the structure and system information in the model depicted by Figure 4.8 and adds further detail by specifying the objects (system elements) that will implement the system, along with their operations (functions) and data exchanges. The classes at the top of the ‘swim lanes’ in the Activity diagram are implemented by one or more of their objects at the top of the so-called lifelines of the Sequence diagram. The flow of actions that snakes horizontally across the diagram in Figure 4.8 and migrates downwards is transformed into a downward vertical sequence of operations along the lifelines in the Sequence diagram. Exchanges between the objects flow horizontally as the events in the environment and the processes of the system unfold.

In the diagram for the ECG in Figure 4.9, two functions and one function call are specified for the implementation of the actions in Figure 4.8. The algorithm for governing emissions is named “determineCalibration ( )”. It is a function of two variables, “Engine State” and “Torque Demand”. The other function is the reporting of the calibrations in a format that is suitable for the engine. In the architecture represented in the diagram, the function call from the ECG to the ADAS is a request for the torque demand that the ADAS will be making to the engine. This method is modular in the sense of Structured Design, in that it does not interfere with the routine operation of the ADAS. The dashed arrow is the message with the response from the ADAS to the ECG. This exchange does not need to be implemented directly between the ECG and ADAS. It could also be implemented through a controller area network (CAN bus). In the Sequence diagram, the interactions from the Use Case diagrams are further defined and resolved by interoperations that can identify interfaces between the elements of the system (which are represented as UML Objects). In practice, the three diagrams are specified iteratively and recursively. Because of the interrelations of the underlying models, it is generally not easy to correctly specify all of these simultaneously in the first attempt.

The interrelational structure of the essential architecture, the system models and the engineering models in Chapter 2 provide the minimal level of information that is needed to specify the ECG system for design and implementation. The architect and systems engineer support these next steps in the engineering process but do not design or implement the system elements. Modern Architecture Definition must also support interface specifications for interoperability as part of system integration. The Sequence diagram in Figure 4.9 does this. It also supports specification of a software architecture.

4.5 SUMMARY AND SYNTHESIS

Relations between form and functionality modelled using the laws of science enable the practice of engineering to deliver reliable products and services with market value. The influence of information technology on engineering has become increasingly significant over the first two decades of the twenty-first century in the commercial sector. This has also been fuelled by the development of large-scale information systems in aerospace and defence such as theatre level mission planning systems in which software, hardware, databases, system operators, and increasingly autonomous vehicles are integrated. The importance of system architecture has therefore become more central to modern engineering, as has the need for more formal approaches to system analysis, design, assurance, and specification. Over the same two-decade time period, advanced systems engineering approaches have been expanding from their origin in aerospace and defence to fulfil a need in the commercial sector to deliver increasingly complex system solutions.

If engineering in general and systems engineering in particular are to be founded upon science, then there needs to be a clear understanding as to what is meant by *science*. The scientific method consists of characterisation of observables (by definition and measurement) and statement of hypotheses that are testable and refutable by comparing predictions from models with experimentation (which must be repeatable). The System Requirements and Architecture Definition processes in Figure 4.1 are similar to the characterisation and modelling processes of science. The Verification and Validation processes are similar to scientific experimentation. It should be noted that measurement always involves interaction with the physical object of interest. A scientific arrangement or method of experimentation may be defined as the gathering of individual objects into a synthetic whole. This concept is very close to that of a system. The object of interest in scientific experimentation is a system or element that is isolated from the environment of the experimentation system based on conditions that are considered to be relevant to the outcome. The use case in Figure 4.10 depicts this arrangement, which has a striking similarity with Use Case diagrams in systems engineering. The only substantive difference is the *description* of objects that already exist in the physical world versus the *intent* of objects that are designed and developed in engineering.

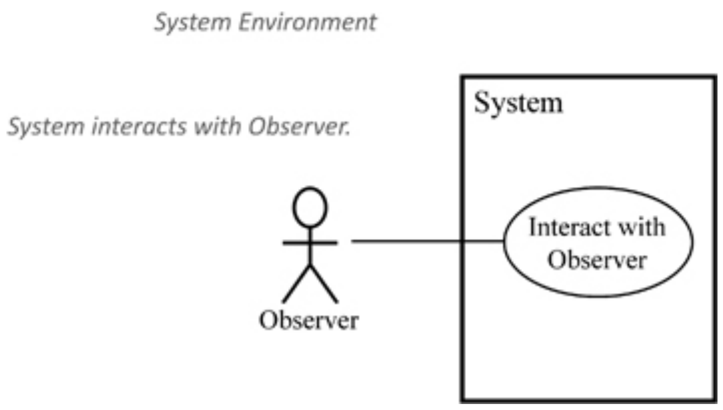

FIGURE 4.10 Use Case diagram for scientific experimentation and discovery

The similarity between science and engineering is an example of logical homology; that is, a congruence that provides guidance for understanding the underlying causes of similarity. Although scientific method is concerned with explanation of observation, homologies have provided useful models in science (Bertalanffy 1969). The Use Case diagram in UML is a logical homology, especially as used in this book in terms of verb-noun predicates. An example in biology is the similarity of cells due to shared ancestry between a pair of structures or genes. In physics, another example is the mathematical similarity of models for heat and electricity as flows of a substance despite their substantial physical differences. Therefore, engineering has more in common with science than just using its laws and models. Logical homology is also used in this book in the similarity of the models in the tutorials in Part II and those of the case studies in Part IV. Over a half century ago, the noted biologist and theorist Ludwig von Bertalanffy considered that any general theory of systems should be able to distinguish between analogies (patterns or other superficial similarities), which are regarded as useless by science, and homologies which, on the other hand, can be useful.

The need for more formal approaches to the architecting and engineering of modern systems demands that concepts and terminology be defined with greater precision, yet remain practical and intuitive for comprehensibility. Formal approaches are a subject of the first-order model theory that underlies the mathematical expression of models in modern science. Modelling is interpretation of concepts into structures that represent the system. This applies to both science and engineering. The definition and lack thereof for key terms in the current relevant international standards generally does not provide the precision required by a rigorous approach to architecture and systems engineering. To fill this gap, essential definitions based on recent summative research (Dickerson et al. 2021) have been offered in Chapter 3 for the terms structure, architecture, system, and engineering. These definitions are used in the essential technical processes and structured methods put forth in this book. They are intended to complement the standards, and not to be a replacement.

The precision and expressive power of the essential definitions are seen in the unified model depicted in Figure 4.11. The synthesis of concepts in the figure extends the logical model of system based on standards in Figure 3.1 of Chapter 3 into the unified model. The six natural language predicates in Figure 3.1 have been synthesised with the predicates of the essential definitions to form a holistic logical model of the eleven natural language predicates depicted in the unified model.

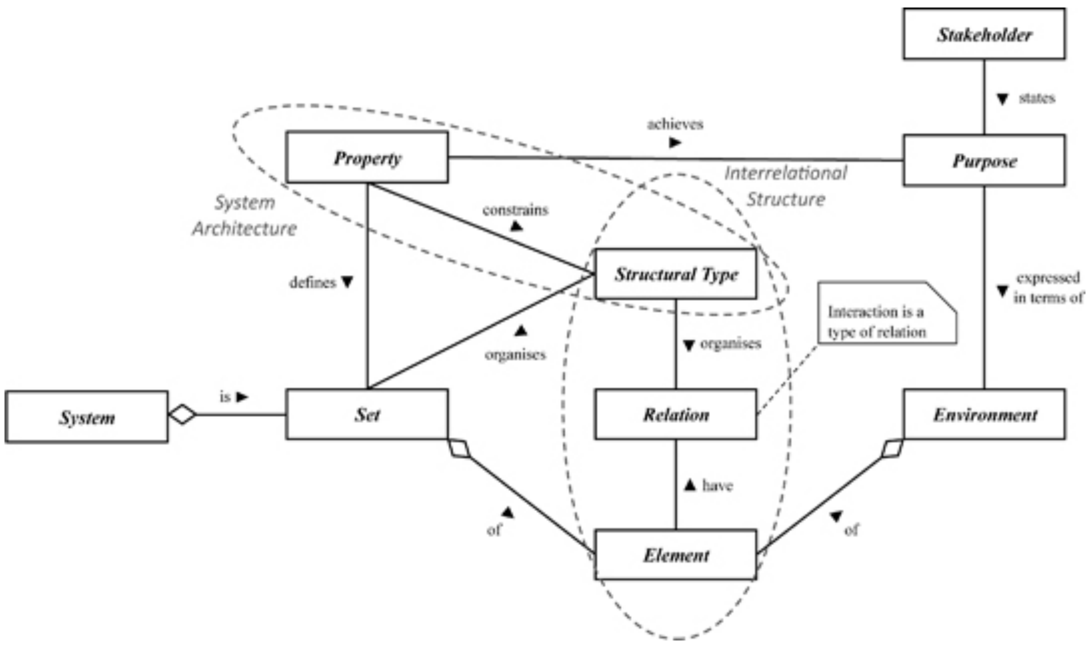

FIGURE 4.11 Synthesis of concepts for system and architecture into a unified model

Figure 4.11 can be used, for example, to explain the relationship and distinction between *system* and *architecture.* The current standards offer no such normalisation of these key terms; and as discussed in Chapter 3, this leads to confusion about the relation between architecture and models in the standards, amongst other things. In the unified model, the definition of system is seen to be a *first-order logical statement* that relates the elements of a defined set of system elements with those of an environment. It can be modelled using only the entities and their first-order relationships in Figure 4.11 (represented by nouns that name abstract classes and the relationship lines between them). On the other hand, the definition of system architecture is seen to be a *second-order logical statement* that relates constraint properties with structural type; it is a relation between properties and relations rather than a relation between entities. (Recall that structure is a relation between the objects of a class.)

This insight has broad implications. One is that the Essential Model of a system produced by the Structured Analysis process and methods in Section 4.2 is indeed a first-order model in the sense of predicate logic and Tarski model theory. Another is that an Essential Architecture of a system produced by the Structured Design process and methods in Section 4.3 specifies an interrelational structure that has been constrained by the Essential Model and various properties such as specified functionalities, performance, constraints from non-functional requirements, and other architectural properties. In general, there is no first-order model of Architecture. This was foreshadowed by the abstract (predicate) representation of the relational framework in Figure 4.5 that underlies the methods of Structured Design used in this book. Recall that in the formal framework, predicates and (logical) sentences are transformed into a predicate structure, i.e., a structure of relations between relations. The reason a Sequence diagram such as in Figure 4.9 can be represented in UML is that UML is not a formal language, despite its roots in class theory and logic. The complexity of the interrelationships in the Essential Architecture will be discussed in the second tutorial (Chapter 6). It should be further noted that the distinction of *system* as a first-order concept in the diagram in Figure 4.11 and *architecture* as a second-order concept also resolves confusions that are a subject of debate in the international community even at the time of the writing of this book.

Chapter 4 has shown how systems, architecting, modelling, and engineering can be bound into a straight-forward process implemented by rigorous methods for architecture and system definition of solutions to complex problems. This fulfils a need for methods that implement an easily accessible yet rigorous end-to-end process to exploit the advances that have been made in tools and languages. This goes beyond traditional system engineering methods. Structured Analysis and Design, for example, have been explained in the context of current standards and modelling languages using a widely accepted core of object-oriented graphical models: Use Case diagrams, Activity diagrams, Class diagrams, and Sequence diagrams. These models express a concept of integration (based on compliance to architecture) for modern systems that goes beyond physical interfaces and connecting parts. Integration today must also include a blueprint for system interoperability. These ideas have been illustrated through the elementary example of the ECG that has been presented in bite-sized steps using a thread of discussion that began in Chapter 2. The mathematical language and methods of science and model theory have been expressed in a straightforward intuitive engineering process using state of the art object-oriented graphical models. The conceptual integrity of the system can therefore be maintained throughout the development process by a comprehensive architecture that enables the design and development of hardware and software in a holistic concordant way.

Part I Bibliography

Bell, John Lane, and Alan B. Slomson. 1969. Models and Ultraproducts. Amsterdam: North Holland.

Brooks Jr Frederick P. 1995. The Mythical Man-month, Boston: Addison-Wesley Longman Publishing.

Cameron, Bruce, and Daniel M. Adsit. 2020. “Model-based systems engineering uptake in engineering practice.” IEEE Transactions on Engineering Management 67, no. 1: 152–162.

Chen, Peter Pin-Shan. 1976. “The entity-relationship model – toward a unified view of data.” ACM Transactions on Database Systems (TODS) 1, no. 1: 9–36.

Cloutier, Robert and Mary Bone. 2015. “MBSE Survey.” Presentation given in INCOSE IW Los Angeles, CA. January 2015. http://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:incose_mbse_ survey_results_initial_report_2015_01_24.pdf

Dickerson, Charles E. 2008. “Towards a logical and scientific foundation for system concepts, principles, and terminology.” In *2008 IEEE International Conference on System of Systems Engineering*, (June 2008): 1–6.

——. 2013. “A relational oriented approach to system of systems assessment of alternatives for data link interoperability.” IEEE Systems Journal 7, no. 4: 549–560.

Dickerson, Charles E, and Dimitri N. Mavris. 2010. Architecture and Principles of Systems Engineering. New York: CRC Press.

——. 2013. “A brief history of models and model based systems engineering and the case for relational orientation.” IEEE Systems Journal 7, no. 4: 581–592.

Dickerson, Charles E, and Siyuan Ji. 2018. “Analysis of the vehicle as a complex system, EPSRC.” Impact 2018, no. 1: 42–44.

Dickerson, Charles E, Siyuan Ji and David Battersby. 2018. Calibration system and method. U. K. Patent GB2555617, filled November 4, 2016, and issued May 09, 2018

Dickerson, Charles E, Stephen M. Soules, Mark R. Sabins, and Philipp H. Charles. 2003. *Using architectures for research, development, and acquisition*. ADA427961. Prepared by Office of the Chief Engineer of the Navy, Assistant Secretary of the Navy. Defense Technical Information Center (www.dtic.mil).

Dickerson, Charles E, Michael Wilkinson, Eugenie Hunsicker, Siyuan Ji, Mole Li, Yves Bernard, Graham Bleakley, and Peter Denno. 2021. “Architecture definition in complex system design using model theory.” IEEE Systems Journal 15, no. 2: 1847–1860. doi: 10.1109/JSYST.2020.2975073.

Dori, Dov. 2016. Model-Based Systems Engineering with OPM and SysML, New York: Springer.

Estefan, Jeff A. 2007. “Survey of model-based systems engineering (MBSE) methodologies.” INCOSE MBSE Focus Group 25, no. 8: 1–12.

Finch, James K. 1951. Engineering and Western Civilization. New York: McGraw Hill.

Hatley, Derek, Peter Hruschka, and Imtiaz Pirbhai. 2000. Process for System Architecture and Requirements Engineering. New York: Dorset House Publishing.

Hawking, Stephen. 1988. A Brief History of Time. New York: Bantam Books.

Hilliard, Richard F., Timothy B. Rice, and Stephen C. Schwarm. 1996. “The architectural metaphor as a foundation for systems engineering.” In INCOSE International Symposium 6 , no. 1: 559–564.

Hodges, Wilfrid. 2020. “Model theory,” In Stanford Encyclopedia of Philosophy, Online ed., edited by Edward N. Zalta. Stanford: Metaphysics Research Lab, Stanford University. https://plato.stanford.edu/entries/model-theory/

IEEE (Institute of Electrical and Electronics Engineers). 2000. *IEEE recommended practice for architectural description for software intensive systems*. IEEE Std 1471-2000.

INCOSE (International Council on Systems Engineering). 2015. Systems Engineering Handbook . 4th ed. New Jersey: John Wiley & Sons.

ISO (International Organization for Standardization). 2001. *System and software engineering – System life cycle process*. ISO/IEC/IEEE 15288:2002. Standard superseded by ISO/IEC /IEEE 15288:2015.

——. 2002. *System and software engineering – System life cycle process*. ISO/IEC/IEEE 15288: 2002. Standard superseded by ISO/IEC/IEEE 15288:2015.

——. 2011. *Systems and software engineering – Architecture description*. ISO/IEC/IEEE 42010: 2011.

——. 2015. *System and software engineering – System life cycle process*. ISO/IEC/IEEE 15288: 2015.

——. 2018. *Information technology – Common Logic (CL) – A framework for a family of logic-based languages*. ISO/IEC 24707:2018.

Klir, George J. 1991. Facets of Systems Science. New York: Plenum Press.

Lin, Tzu-Chi, Siyuan Ji, Charles E. Dickerson, and David Battersby. 2018. “Coordinated control architecture for motion management in ADAS systems.” IEEE/CAA Journal of Automatica Sinica 5, no. 2: 432–444.

Lin, Yi. 1999. General Systems Theory: A Mathematical Approach. New York: Kluwer Academic/Plenum Publishers.

Lin, Yi, and Yong-Hao Ma. 1987. “Remarks on analogy between systems.” International Journal of General System 13, no. 2: 135–141.

OMG (Object Management Group). 2003. *MDA Guide Version 1.0.1*. Edited by Joaquin Miller and Jishnu Mukerji. OMG document omg/2003-06-01. OMG

——. 2014. *MDA Guide Version 2.0*. OMG document omrsc/2014-06-01. OMG

——. 2017a. *OMG Systems Modeling Language (SysML®)*, Version 1.5.

——. 2017b. *Unified Modeling Language (UML®)*, Version 2.5.1.

——. 2019. *UML Profile for MARTE (MARTE)*, Version 1.2.

Pyster, Art, Rick Adcock, Mark Ardis, Rob Cloutier, Devanandham Henry, Linda Laird, Michael Pennotti, Kevin Sullivan, and Jon Wade. 2015. "Exploring the relationship between systems engineering and software engineering." Procedia Computer Science 44, no. 2015: 708–717.

Rosen, Robert. 1993. "On models and modeling." Applied Mathematics and Computation 56, no. 2–3: 359–372.

Sowa, John F. 1983. Conceptual Structures: Information Processing in Mind and Machine. Reading: Addison-Wesley.

——. 2000. Knowledge Representation: Logical, Philosophical, and Computational Foundations . Brooks: Cole Pacific Grove.

Tarski, Alfred. 1954. "Contributions to the theory of models. I & II." Indagationes Mathematicae (Proceedings) 16, no. 1954: 572–588.

——. 1955. "Contributions to the theory of models. III." Indagationes Mathematicae (Proceedings) 17, no. 1955: 56–64.

Von Bertalanffy, Ludwig. 1967. "General theory of systems: Application to psychology." Social Science Information 6, no. 6: 125–136.

——. 1969. General Systems Theory. New York: George Braziller Inc.

Wasson, Charles S. 2005. System Analysis, Design, and Development: Concepts, Principles, and Practices. Hoboken: John Wiley & Sons.

Wymore, A. Wayne. 1993. Model-based Systems Engineering. Boca Raton: CRC Press.

Yourdon, Edward. 1989. Modern Structured Analysis. Upper Saddle River: Yourdon Press.