# Architecture Definition in Complex System Design Using Model Theory

Charles E. Dickerson, *Senior Member, IEEE*, Michael Wilkinson, Eugenie Hunsicker, Siyuan Ji, *Member, IEEE*, Mole Li, *Member, IEEE*, Yves Bernard, Graham Bleakley, and Peter Denno

***Abstract*—Architecture definition, which is central to system design, is one of the two most used technical processes in the practice of model-based systems engineering. In this article, a fundamental approach to architecture definition is presented and demonstrated. The success of its application to engineering problems depends on a precise but practical definition of the term architecture. In the standard for architecture description, ISO/IEC/IEEE 42010:2011, a definition was adopted that has been subsumed into later standards. In 2018, the working group JTC1/SC7/WG42 on system architecture began a review of the adopted definition, holding sessions late in the year. This article extends and complements a position paper submitted during the meetings, in which Tarski model theory and ISO/IEC 24707:2018 (logic-based languages) were used to better understand relationships between system models and concepts related to architecture. Independent from the working group, it now contributes intuitive fundamental definitions of the terms architecture and system that are used to specify a mathematically based technical process for architecture definition. The engineering utility and benefits to complex system design are demonstrated in a diesel engine emissions reduction case study.**

***Index Terms*—Architecture, category theory, definition process, diesel emissions, model theory, system.**

## I. Introduction

ARCHITECTURE is key to the modern practice of engineering but in many ways, a precise practical definition has been elusive if not ineffable. The term would be understood by a general audience as a property of buildings or large-scale structures. Although understanding architecture in this way is intuitive and useful, it lacks the precision needed for application to engineering problems. The position taken in this article is the same as the one taken with JTC1/SC7/WG42 in late 2018: a prose definition of a technical term should be complemented by a mathematical interpretation [1]. This is a model-theoretic method for assessing validity of definitions that is essential for specifying an architecture definition technical process for engineering and scientific problem solving.

### A. Ubiquity of the Architecture Metaphor

In civil engineering, architecture relates a building's purpose (function), form (how its spaces are organized to achieve the purpose), and construction (what it is built from and how it is built). Beyond the design and construction of buildings, architecture has found a place in many other different disciplines, serving as a readily apprehended metaphor by which the native "structures" of a subject can be understood.

Although architectural ideas are now prevalent in disciplines as diverse as civil engineering, systems engineering, management science, biology, and mathematics, there is no consensus on terminology or meaning—there is common ground but there is no unity. The potential of achieving precise unified definitions by means of a formal approach and the benefits of so doing have been long recognized in, for example: the foundational work of Bertalanffy [2], which was inclusive of many mathematical expressions of systems concepts, Wymore's codification of model-based systems engineering (MBSE) [3], which expressed a program for systems engineering, and Rosen [4], who was perhaps the first to recognize the possibility of using category theory for systems and scientific problem solving. The authors have also long recognized and investigated formal approaches in: using architecture to develop and acquire mission-level capabilities [5], interpretation of an adopted definition of "system" as a Hamiltonian system in physics [6], and more recently the Wilkinson polemic paper [7]. In a shift from a narrow systems orientation, recent efforts within JTC1/SC7/WG42 have applied the term architecture to entities not normally considered to be systems. Note, however, that the purpose of this article is neither to report on such work nor to critique it. Rather the purpose is to offer definitions of architecture and system that are complemented by mathematical interpretation that can be used for specification of a technical process for architecture definition and can be exploited in engineering practice and in relevant standards.

Manuscript received December 4, 2019; revised February 1, 2020; accepted February 15, 2020. Date of publication March 4, 2020; date of current version June 16, 2021. *(Corresponding author: Charles E. Dickerson.)*

Charles E. Dickerson is with the Wolfson School of Mechanical, Electrical and Manufacturing Engineering, Loughborough University, Loughborough LE11 3TU, U.K. (e-mail: c.dickerson@lboro.ac.uk).

Michael Wilkinson is with the BAE Systems Maritime, Barrow-in-Furness LA14 1AF, U.K. (e-mail: michael.wilkinson1@baesystems.com).

Eugenie Hunsicker is with the School of Science, Loughborough University, Loughborough LE11 3TU, U.K. (e-mail: e.hunsicker@lboro.ac.uk).

Siyuan Ji is with the Department of Computer Science, University of York, York YO10 5GH, U.K. (e-mail: s.ji@lboro.ac.uk).

Mole Li is with the Rolls-Royce, Derby DE24 8BJ, U.K. (e-mail: mole.li@rolls-royce.com).

Yves Bernard is with the Airbus Defence and Space, 31400 Toulouse, France (e-mail: yves.bernard@airbus.com).

Graham Bleakley is with the IBM UK, Ltd., Portsmouth PO6 3AU, U.K. (e-mail: graham.bleakley@uk.ibm.com).

Peter Denno is with the National Institute of Standards and Technology, Gaithersburg, MD 20899 USA (e-mail: podenno@gmail.com).

Digital Object Identifier 10.1109/JSYST.2020.2975073

## B. *Precise Natural Language Definitions*

The ambiguities introduced by using natural language in the definition of technical terms can be reduced and possibly resolved by introducing languages that are more precise or formal; the predicate calculus of logic being just one. An issue though is that such languages may not be accessible to a general audience or to the stakeholders in a discipline.

A pioneer of digital computer architecture, Brooks [8], proposed a resolution to this issue that remains valid today

*One needs both a formal definition of a design, for precision, and a prose definition for comprehensibility.*

This idea was adopted as a "principle of definition" by Dickerson and Mavris [9] and will be adapted to reason about the terminology of architecture and systems. The term "*a formal definition of a design*" will be replaced by "*a mathematical interpretation of a concept.*" Concordance between the two types of definition is achieved by interpreting natural language (prose) into mathematical logic and models. The primary challenge in this approach is to preserve concordance between precision and comprehensibility without losing either.

The position paper [1] put forward to the working group JTC1/SC7/WG42 applied this principle to the definition of (system) architecture in [10]. This definition will be referred to as the "adopted" definition. As mentioned earlier, recent work within WG42 has generalized the adopted definition to refer to entities other than systems.

It was argued in the position paper that there is a need for distinction between key terms such as: *concept, property, embodiment, element, and relation*. Based on Tarski model theory [11], [12], [42] an initial attempt to refine the wording of the adopted definition was made to illustrate how a better distinction could be achieved. Terminology for *object, property*, *class, and type* from mathematics will now also be used for further precision and clarity. For example, *class* is a technical term in set theory for defining a collection of objects that share one or more common properties.

Fundamental definitions are now offered independent of the working group that can be used to make a distinction between key terms and express an intimate relation between architecture and structure. The first definition proposed is as follows.

*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 that of the adopted definition of architecture, but it 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*.

One mathematical interpretation of the proposed definition of structure is the *separation* of the collection of the counting numbers $\mathbb{N}$ into two disjoint subsets based on the *property* of divisibility by the number 2. One of the subsets is comprised of the even numbers and the other the odd numbers. The *junction* (union) of the two subsets comprises the whole of $\mathbb{N}$. This type of structure in mathematics is called a partition. It separates the underlying set into equivalence classes. Partitions of a collection into equivalence classes are common in abstract algebra and more generally in category theory [13]. Thus, structure as defined in this article adheres to the principle of definition.

The mathematical term *object* is used in the definition because it is an abstraction that is only constrained by the properties it possesses, e.g., functional, physical, or temporal. Note that *elements* (of a set) are mathematical objects. "Elements" is a usage of terms in alignment with the adopted definition of architecture. However, it should be noted that although the term structure is used widely in the engineering community, there is no common definition offered in current relevant standards.

In specific domains, such as software engineering, a term, such as *stable binding,* is often used instead of junction. Thus, the proposed definition of structure accommodates ideas, such as stable bindings of static objects, instantaneous bindings (events), as well as dynamic behavior. It is also worth noting that most definitions of structure only mention some form of junction, but they are silent on separation. However, these two terms should be on an equal footing. For example, when a system boundary is defined, a collection of objects of interest is *separated* into members of a system and those of an environment.

Structure therefore expresses a relation among the objects of a collection. The general character of this relation is a specialized property, which is referred to as a *type*. In software engineering, the term *classifier* is often used instead of type. This leads to the second definition, which is as follows.

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

Such properties will be called *architectural properties*. Note then that every architecture is associated with at least one architectural property, its structural type. Architectural properties can constrain and further specify structure. An example was investigated in detail in the position paper [1]. In engineering practice, architecture can be considered as a qualified structural type, i.e., one with qualities (e.g., implementable properties) that can achieve a defined purpose.

The association of a structural type with a collection of properties in the proposed definition of *architecture* can then be represented as an ordered pair. This construct can also provide a formal underpinning to architecture frameworks, which are commonly visualized as a matrix. Each row can be defined by one or more architectural properties and each column can be a structural type, with the intersection (a cell in the matrix) defining a class of architecture implemented by the architectural properties in a way that is consistent with the structural type. An example is when the functionality of a system is implemented in the graphical structure of a use case in object-oriented modeling.

The fundamental definitions offered in this article will be referred to as "essential definitions." They are subsistent in the sense of being an economical choice of generally understandable words that have mathematical interpretation. The essential definition of architecture is complemented as follows.

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

This definition is adapted from a mathematically based one cited by Bertalanffy [2] who has used the term *interrelated*, which involves *relations among relations*. Also, the environment in his definition is on an equal footing with the whole (set of elements), but its relation to the elements is not specified. It is not necessary for the set as a whole to belong to the environment or even for it to interact with it. The definition though does not consider a system without making reference to an environment.

When a structure of interrelations has been identified or defined, the set is said to be endowed with an interrelational structure (Note: mathematicians use the term *endowed* for the pairing of a set with a relational structure.). Thus, a system is a set of elements endowed with a structure that is of an interrelational type. In this sense, system is a realization of architecture. It will also be seen that the interrelational structure of a system can be used to specify architecture. This provides a mathematical basis for a process that will be called essential architecture definition. The term is used in the sense of Yourdon structured analysis in which "essential" means necessary or essence.

The essential definitions in this article underlie those specified by ISO in [10] and [14]. Note, for example, that the concept of "a set of interrelated elements" includes that of "a combination of interacting elements." There are numerous other definitions in relevant standards. Most consider the commercial and life-cycle aspects of engineering, such as definition and management of the system configuration, translation of system definition into work breakdown structures, among others. The essential definitions are intended to complement the standards, not to be a replacement.

### C. Structure of this Article

The article is structured around three key contributions: the essential definitions of the terms architecture and system, their interpretation into a mathematically specified technical process for architecture definition, and a detailed demonstration of engineering utility and benefits to complex system design. Section I has established the problem to be addressed and the context and offered "essential definitions." Sections II and III provide a historical and theoretical background. Section II is a brief history of relevant definitions of architecture and significant points about their evolution. It also has a brief summary of early attempts at system architecting (which is another name for architecture definition), its relation to system engineering, and issues associated with the definition of terms. Section III provides an explanation of conceptual structures and Tarski model theory that should be accessible to a general audience. It concludes with a comparative analysis of the adopted definition of architecture and the essential definition. These will be seen to be in general agreement. Some ambiguities in the adopted definition will also be addressed.

Section IV builds on these foundations to show how an architecture definition process that supports system design can be implemented. This provides a demonstration of the validity of the essential definitions offered in Section I-B. Section V applies the process to the specification of a calibration system for diesel engine emissions reduction. Constraint-driven design (CDD) methods are applied to the calibration problem. This demonstrates the engineering utility of the proposed architecture definition process. Section V concludes with how the process has been used in an MBSE standard concerned with CDD.

## II. Brief History of Architecture

It is instructive to understand how the definitions of architecture and related terms in systems engineering have evolved over time and how they have been influenced by other disciplines. Although this might seem to be only of theoretical interest, the incorporation of such definitions within key standards has a significant impact on how systems engineering is conducted in practice. This section begins with a brief summary of the diversity of meanings of systems architecting and moves on to describe how key definitions have evolved in canonical standards. The model-driven architecture (MDA) in object-oriented software standards provides an initial example of how the essential definition can be used in an explanatory way by refining the usage of the term architecture in software development. It concludes with a short section on architecture styles and patterns.

### A. System Architecting

A detailed examination of the nature and impact of different meanings of system architecting was conducted by Emes *et al.* in 2012 [15]. This work reported on research to capture and analyze beliefs about the meaning of architecting and related terms. It considered definitions from three authoritative sources, including ISO/IEC/IEEE 42010:2011 [10], and from a set of interviews with practitioners. Twelve contentious questions about architecting were investigated and six different perspectives on the meaning of system architecting and its relationship to system engineering were described.

The most emphatic conclusion from the research was that a consensus on the meaning of architecture did not exist and there were many diverse interpretations. The value of explicit formal definitions, such as those found in standards documents, was therefore judged to be particularly significant. This article noted that the lack of robust definitions for key terms made the development of standards more difficult. It also noted that its scope excluded systems of systems and enterprises, for which further diverse interpretations of key terms might be expected.

More recently, ISO/IEC/IEEE 42020 [16] has defined a standardized approach to architecting, including processes for architecture conceptualization and elaboration. ISO/IEC/IEEE 42030 [17] has defined a standard for architecture evaluation. It is too early to judge what impact these new standards will have but, over time, they are likely to influence understanding of architecting and the definitions in other standards. These latest standards make use of the same definition of architecture, as described in Section II–B and discussed in Section III–D.

### B. System Architecture

The earliest attempts to standardize ideas about the meaning of architecture within systems engineering were in connection with the lifecycle processes standard ISO/IEC 15288. The first published version (2002) was influenced by legacy software engineering standards, which conditioned the way in which architecture was conceived and also combined it with design into a single process called architectural design. This was separated into two processes in the 2015 update [14].

The software discipline had additional influences. In the early 1990s, 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 relationships). In 1996, with the aim of using an architectural metaphor as a foundation for systems engineering, Hilliard *et al.* [18] 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 [19], 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, [20] for example, considers architecture as a graphical model (or representation).

In the latest published versions of the key standards, specifically ISO/IEC/IEEE 42010:2011 [10] and 15288:2015 [14],[1] the adopted definition of architecture refers to "fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution." Here, the earlier term "organization" has been replaced by "concepts or properties." Rather than organization being embodied in a static structure (elements and their relations), it is the concepts and properties that are embodied.

A slightly refined definition has been adopted in [16] and [17], in which the idea of embodiment is now referred to only in a note. In a further refinement, these standards associate architecture with "architecture entity" rather than "system." Finally, echoing many of the aforementioned points, the practitioner's view of architecture, as expressed in the Systems Engineering Book of Knowledge [21], also associates architecture with abstraction, conceptualization, and high-level structure.

## C. Model-Driven Architecture

Over the same two-decade time period, the Object Management Group (OMG) has used architecture in standards for software development. For example, MDA is an approach to software design, development, and implementation. MDA provides guidelines for structuring software specifications that are expressed as models [22]. In MDA, the software architecture of an application is the basis for deploying the computer code.

MDA development focuses first on the functionality and behavior 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 behavior are modeled once and only once. The PIM is implemented by one or more platform-specific models (PSMs) and sets of interface definitions until 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. In the separation of platform technology from functionality and behavior, MDA conforms to the precepts of structured analysis as put forth by Yourdon.

In MDA, the models of functionality and behavior in the PIM are transformed into PSM models by associating new concepts not found in the PIM. The models of the PIM are thus joined by transformation with those of the PSM. This collection of models therefore has both junction and separation that express a defining property of the collection, i.e., an approach for software design, development, and implementation. This is the definition of *structure*. Furthermore, because the PIM and PSM are defined as *types* of models, this structure derives from a structural type. Consequently, in terms of the essential definitions of this article, MDA can then be considered as an architecture *vis-à-vis* an approach.

## D. Architecture Levels, Styles, and Patterns

It is common within systems engineering practice guidelines and textbooks to identify an iterative or "fractal" process structure, including architecture definition, at each of the process steps. From this perspective, architecture is developed at each level in the hierarchy of system levels of detail. Within the well-known NASA Systems Engineering Handbook [23], for example, the systems engineering process at each level includes five distinct steps (four definition processes and one implementation process) defining context, requirements, architecture, and design, then proceed to implementation if the lowest level of detail is reached, otherwise return to step context. This kind of thinking can be made harmonious with the standards definitions, as long as the "entity" at each level being architected is considered as a system in its own right.

Finally, architectural styles and patterns should also be mentioned. Like architecture itself, these are problematic terms with no universally accepted definitions. Most authors seem to use "style" as shorthand to codify a correspondence between different architectures, whereas "pattern" is a reusable style that addresses a defined type of problem.

## III. Using Tarski Model Theory

A general audience would understand the term "model" as a representation of something or an excellent example [24]. In order to avoid confusion between models and architecture, a well-established definition of "model" from mathematics and logic is reviewed and used to complement natural language definitions. In mathematics, diagrams or symbolic expressions are generally used to *represent* models. In this section, the term "concept," which is central to the definitions of architecture reviewed in Section II, will be understood in terms of

[1]Permission to reproduce extracts from British standards is granted by BSI Standards Limited. No other use of this material is permitted (For the convenience of readers, the equivalent international reference is cited.).

conceptual structures that can become models when interpreted mathematically.

## A. Motivation From Algebra

The concept of orthogonality (right angles) in the plane geometry of triangles can be represented algebraically. If *a, b*, and *c* are the lengths of the sides of a triangle, and if these lengths have the algebraic relation $a^2 + b^2 = c^2$, then the triangle is said to be a right triangle. Additionally, the two sides corresponding to $a$ and $b$ are said to form a right angle. This relation is an algebraic sentence that expresses a concept of orthogonality. Model theory is simply an abstraction of the idea that models are sentences interpreted into relational structures. The sentences can express theories or concepts.

## B. From Algebra to Logic: Model Theory

The propositional calculus and the predicate calculus are languages of mathematical logic. Propositions are declarative statements represented by variables that take on the value of either true or false. This language of logic is inadequate for a proper theory of models.

Predicates, on the other hand, are declarative statements of relationships between variables and are represented by predicate letters $\{P_1, P_2 \ldots\}$ that represent relations among individual variables $\{v_1, v_2 \ldots\}$ as well as constants $\{c_1, c_2 \ldots\}$. A *sentence* in the predicate calculus is defined to be a fully quantified well-formed formula. The predicate calculus (with equality) is the level of precision in language necessary for a mathematical theory of models.

Tarski model theory 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 of relations $\{R_\alpha\}$ on $M$.

Thus, the algebraic expression of orthogonality in the Tarski theory corresponds to the predicate sentence given by

$$\forall v_1, v_2, v_3 : P_1 (v_1, v_2) = P_2 (v_3) . \tag{1}$$

Let the set *M* be the Cartesian plane. The interpretation of the predicates into relations assigns $P_1(v_1, v_2)$ to $a^2 + b^2$ and $P_2(v_3)$ to $c^2$, for specific values of *a, b,* and *c* that are lengths of the sides of a triangle (e.g., 3, 4, and 5). Every triangle $\Delta$ in the Cartesian plane has a relation $\mathrm{R}_\Delta$ among its sides. When the relation conforms to $a^2 + b^2 = c^2$ then the predicate sentence is valid, and the triangle is a model of orthogonality. A more detailed mathematical review of the predicate calculus (with equality) and Tarski model theory can be found in [25]. A general but technical review for the nonspecialist is given by the model theorist Wilfrid Hodges in the Stanford Encyclopedia of Philosophy [26].

## C. Comparison With Conceptual Structures

The framework of conceptual structures described by Sowa [27] is essentially an approach to representing knowledge inside computer systems, but it is more general and can be applied more widely. In this framework, a generic concept is a mental interpretation of a percept (which is a correlation of sensory impressions of the physical world). In this case, the concept is called concrete. Concepts that have no such percepts are called abstract. Sowa points out that, although concrete concepts may have direct associations with sensory impressions (percepts), this is not the case for abstract concepts, which are however no less meaningful. In other words, every thought that is effable (capable of being expressed) can be regarded as a concept.

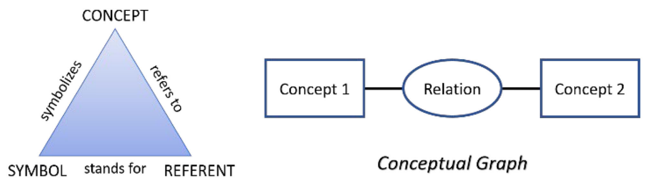

Fig. 1. Meaning triangle and concepts.

An abstract syntax and model-theoretic semantics for conceptual graphs are provided in [28]. Within the Sowa approach, concepts are related to symbols and to referents in a “meaning triangle” depicted in Fig. 1 [27] and based on Ogden and Richards [29].

Here, a mental CONCEPT is symbolized by SYMBOL, which could be a word. The extension of the SYMBOL is all the things it stands for—which could be thought of as instances of the CONCEPT. In other words, CONCEPT is an intentional abstraction of the common properties of the REFERENTs to which it refers. According to Sowa [30], every concept has a *type,* which defines a class of referents. The *type* is therefore a specification for a set or class of entities in some domain. The type specification is a monadic predicate, e.g., in logic, $P(v)$.

The meaning triangle can be given a mathematical interpretation based on first-order model theory. To say that a symbol “symbolizes” a concept means that SYMBOL gives CONCEPT an identity, e.g., a name or a tag. In the model theoretic semantics of conceptual graphs, CONCEPT has a type, which defines a class of REFERENTs. The association of SYMBOL with REFERENT in the triangle is an instance of what is called an *interpretation map* in the mathematical theory of models.

In a sentence expressed in a language (L) about one or more concepts (as in the conceptual graph in Fig. 1), each concept will have an associated meaning triangle. The sentence applies to every referent that belongs to the class and to the symbol of the concept. The symbols in the sentence belong to what are collectively called the *signature* ($\sigma$) of the language (L).

A model of the sentence involves two things: a choice of referents and the interpretation of the signature in a way that results in the sentence being true (i.e., the interpretation satisfies the sentence). Note that the interpretation map is specified at the level of symbols rather than the level of the entire sentence.

Conceptual graphs are symbolic expressions that represent relations between concepts or their referents. The elementary conceptual graph as depicted in Fig. 1 without any indication of directionality is intended to be read from left to right for ease of expression in natural language. In this case, “Concept 1 has relation to Concept 2." Sowa uses the following notation:

$$[\text{Concept 1}] \rightarrow (\text{Conceptual Relation } 1 \rightarrow 2) \rightarrow [\text{Concept 2}]\,. \tag{2}$$

This is interpreted as meaning that a concept [Concept 1] is related to another concept [Concept 2] via a conceptual relation (Conceptual Relation $1 \rightarrow 2$). Concepts and conceptual relations are further defined in terms of their types and any constraints applying to them. This is equivalent to having limited knowledge captured in a domain ontology.

To represent the conceptual relation in Fig. 1 using the language of the predicate calculus, each of the types associated with the concepts needs to define a set of referents. The model theoretic semantics for this graph is then stated in terms of the referents (entities), $\exists v_1, v_2 : P(v_1, v_2)$ where $v_1$ and $v_2$ take on the values "Entity 1" and "Entity 2." Each entity is a referent of the respective concept, and the predicate letter *P t*akes on the value "Relation." Higher order relations may be required for practical applications and are admissible in conceptual structures.

The above-mentioned text is an example of the "principle" prescribed by Brooks. However, it will be seen in Section V that not every concept can be represented in the model theoretic semantics of a conceptual graph. This is due to fundamental limitations in the predicate calculus.

According to Sowa, the meaning of any concept is acquired solely through its contextual relations as part of a subjective "semantic network," whose nodes include other concepts, percepts, motor mechanisms, emotions, and other mental phenomena. In other words, what gives a conceptual graph meaning is its interpretation via a mind's semantic network. This echoes a subjective view of architecture described in Section II.

Some of the semantic network can be formalized in a way that makes its internal workings at least partly explicit. This "explication of the implicit" can be thought of as the manifestation of concepts and conceptual relations (as brought together in conceptual graphs). Making concepts and their relations explicit allows logical reasoning to be applied—which is the basis of mathematics, science, and engineering. The syntax and model theoretic semantics of conceptual graphs provide a first-order language that can be used to implement the essential architecture definition process to be introduced in Section IV.

### *D. Analysis of Distinction Between Key Terms*

This section concludes with a consideration of everything presented in this article up through Section III–C for the purpose of assessing the distinction between architecture and key terms, such as *property, concept, embodiment, element*, and *relation*. *Class* will also be addressed although it was not one of the key terms in the adopted definition.

It is important to keep in mind that the essential definition of architecture makes no direct reference to either of the terms *element* or *relation*. Nonetheless, the term *structure,* as defined in Section I–B, can be interpreted as a relational structure in model theory, which is in fact comprised of elements and relations. Therefore, these two terms will be considered together as (relational) structure. The essential definition also makes no reference to the term system, and hence, it has no inherent dependence on its definition.

For the essential definition proposed in this article, architecture is a property but not every property is architectural. This is because type is a specialization of property, and architecture has been defined as structural type. Similarly, architecture is not a concept per se. When expressed in the syntax and model theoretic semantics of knowledge representation, every concept has a type. Concepts that have a structural type can be architectural. This is consistent with the adopted definition. It should also be noted that architecture is not conceptualization. Some of the 1990s definitions considered it to be, but it would be better to regard conceptualization as an activity in an architecture definition process in order to avoid confusion.

Furthermore, architecture is not a structure. Rather, it is a structural type. Because type is a specialization of property, and properties define a class or set but are distinct from what they define, architecture is not a class and is also not a conceptual structure. The distinction between models and architecture is then clear. Models are sentences interpreted into structures. (The sentences express theories or concepts.) Consequently, if architecture is a model, it is then a structure, which as previously noted is not the case.

Finally, *architecture is <u>not</u> embodiment into elements and relations*. The essential definition associates structural type with architectural properties that can be implemented by that type. In some respects, this is similar to the idea of "embodiment." However, the embodiment of a property (or concept) into elements and relations (i.e., a relational structure) would be an architectural model (in the sense of Tarski). Thus, the use of the term embodiment in the adopted definition, including the refinement in [16] and [17] mentioned in Section II–B, risks confusion between architecture and model. This can be an issue in practice with the adopted definition for architecture description.

In summary, models are interpretations of sentences (e.g., theories or concepts) into structures. Architecture defines the type of structure to be used as well as further properties that must be satisfied (e.g., constraints and other relations), but is not itself the structure. An architectural class is an implementation of the type and properties. Given an object of interest, architecture is then the basis for constructing models that describe or define the object. The mathematical foundation of the essential definition provides distinction between key terms related to architecture.

## IV. Architecture Definition and System Design

As formalized in this section, the essential architecture definition process involves: defining theories and concepts relevant to an interrelational structure, specifying the symbols for the language associated with the theories, and specifying fully interpreted models of the theories. Using this process, the precepts of MDA can be extended to engineering in a straightforward way for general system design and not just software development. At this level of abstraction, the process can be applied broadly to the life-cycle specification of a system.

## A. Architecture Definition and MDA

Essential architecture definition is at a higher level of abstraction than the processes in currently adopted standards. Based on the first-order model theory of Section III, it can be understood as follows in a mathematically rigorous way.

1) Define a sequence of theories $\{T_j\}_{j=1,\ldots,n}$ relevant to an interrelational structure $(A = R_0, R_1, \ldots, R_n)$ described in natural or technical language relevant to a conceptual graph that formalizes a concept.
2) Specify a sequence of signatures $\{\sigma_j\}_{j=1,\ldots,n}$ for the language(s) associated with the theories.
3) Specify fully interpreted first-order models using the interpretation of the theories $\{T_j\}_{j=1,\ldots,n}$ in terms of the relational structures $\{(R_{j-1}, R_j)\}_{j=1,\ldots,n}$.

In model theory, a *theory* is a consistent set of sentences in a language appropriate for the types of structure considered [25].

When an OMG MDA PIM is implemented for a software system using unified modeling language (UML), the functionality of the system can be specified by means of use case diagrams and behavior of the system through activity diagrams. The specification of functionality in a use case is an example of pairing a property with a structural type that can be implemented by a class. The use cases can also be represented in terms of conceptual structures. Functionality is therefore implemented in two classes of structure of that same type. Each class will be useful for engineering but in different ways. One will be a class of conceptual structure and the other will be an object-oriented class (a use case) that can be used in the software PIM. One type of sentence relevant to a use case is a statement of functional requirements. The quantitative constraints to which the system must comply will be captured through interpretation of the conceptual structure. The use of *common sentences* for functional requirements interpreted into these structures (which are of the same type) will result in equivalent models and assure architecture concordance. The sentences are *theories*. When they refer to a system, they are concepts about the system.

The use cases will form a relational structure that will have interrelations when the system is fully specified. The general form of the type of use case for representing functionality is a verb—noun phrase that associates an actor that interacts with the system by means of the use case. Because the verb establishes a relation between the system and actor, the phrase is a monadic predicate and therefore is a structural type. This type of use case can be represented as an interpretation of the sentence, “system interacts with actor.” The verb phrase is “interacts with” and the actor is the noun. The structure represented by a use case diagram is a *partially interpreted model* of the sentence. In this functional representation of the system, elements of the system are not yet specified, rather only functional relations are.

A so-called graphical model, such as a use case diagram, represents a mathematical relational structure of vertices and edges [31] into which sentences have been interpreted. The vertices are sometimes called the nodes of the graph. The nodes and edges are *symbols in the signature* of the model-based language (e.g., UML) in which the theories have been expressed. Typical symbols for the system (with its boundary) and the actor are a rectangle [which contains the name of the system and the use case(s)] and a stick figure (which is given the name of the actor). These are the nodes. The edge is a line that represents an (interaction) association between the system and actor.

A conceptual graph on the other hand depicts a relation among entities. Each node in a conceptual graph is either an entity or a relation. The edges in the graph serve only to associate the nodes but otherwise have no meaning. The conceptual graph is an interpretation of a sentence into a relational structure, hence it is a representation of a model.

Therefore, graphical models of functionality represent models of a functional architecture for a system. This application of essential architecture definition is in agreement with what is meant by an MDA PIM. It is also an example of how it can be applied broadly to a life-cycle specification of a system.

## B. Extension of MDA to Model-Driven System Design

Essential architecture definition supports an extension of MDA to design and development that encompasses both hardware and software. The nature of engineering considered in [9] is based on its evolution in western civilization [32]. Engineering of a product begins with craft but must be honed using the laws of science for the precision and repeatability needed in commercial production. The following definition aligns well with the essential definitions that have been proposed.

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

The architectural viewpoint in this definition is worth noting. The association of structure and property (i.e., functionality) conforms to the pairing in the essential definition of architecture. Also, rather than focus on functionality and behavior, as MDA does with the PIM, the emphasis is on structure and functionality. In the higher level of abstraction of the essential definitions, behavior is a type of structure and functionality is an architectural property. As with other terms defined in this article, there are numerous definitions of engineering in multiple relevant standards, e.g., [14] and [33].

In this definition of engineering, relations between the structure and functionality of a system are the basis for concept realization. The centrality of the term “concept” to both architecture and engineering makes possible the definition of a process for system development that mirrors the underlying process of MDA for software development. This will be termed as model-driven system design (MDSD) and the process is depicted in Fig. 2. It is based on four types of models: concept, design objectives, design space, and design specification. As depicted in the figure, these models are linked by quantification, model transformation, analysis and optimization, and assurance. MDSD is not intended to be an end-to-end life-cycle process for system development.

Requirements can be regarded as concepts about the system and also linked with system architecture. In this way, requirements and other concepts about the system can be used as a starting point for MDSD. It should be noted that requirements

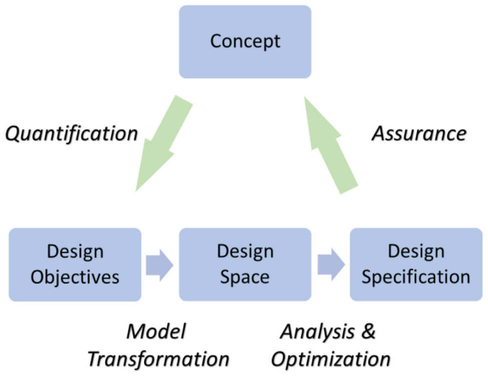

Fig. 2. Model-driven system design.

definition is one of the two most used technical processes in the practice of MBSE. Architecture definition is the other [34].

In modern software and engineering practice, the architecture of a system is generally considered to be the first artifact of design [21]. For MDSD, the essential architecture definition process can be applied to but is not limited to concept definition and specification of design space architecture, the models of which become the subject of analysis and optimization for design specification. Thus, the outcomes of architecture definition as conceived in this article can include the early artifacts of system design but are not limited to these.

## C. Constraint-Driven Design

The engineering and design of complex systems can be driven by constraints on design objectives, such as stakeholder requirements or compliance to regulations. This type of constraint can impose limitations on the design space. CDD is a system design approach in which design objective constraints are a central concern. A solution obtained from such a design approach aims to satisfy all design objective constraints [35].

CDD conforms to MDSD in a mathematically natural way. When the system requirements (to include design objective constraints) are specified using conceptual structures, the quantification of concepts into design objectives can result in mathematical relations. For a given design objective $z$, the simplest type of constraint is in the form $z \leq c$. This could be, for example, a mathematical interpretation of the concept "the measurement $z$ of a specified emission must be less than or equal to a regulatory constraint $c$."

The design space typically comprises several design variables (which are sometimes referred to as factors). For each variable $x$, constraints of the form $a \leq x \leq b$ are imposed. This can be based on domain knowledge, for example. Each constrained 1-D factor of the design space will be in the form of such an interval. As a concept, design variable constraints are represented graphically in Fig. 3. The bounded regions illustrate the structure of the relational-oriented systems engineering technology tradeoff and analysis (ROSETTA) matrix representation depicted in Fig. 9 and explained in Section V–C.

In a 2-D design space, constraints of this type on each factor will be in the form of a rectangle, which is referred to as

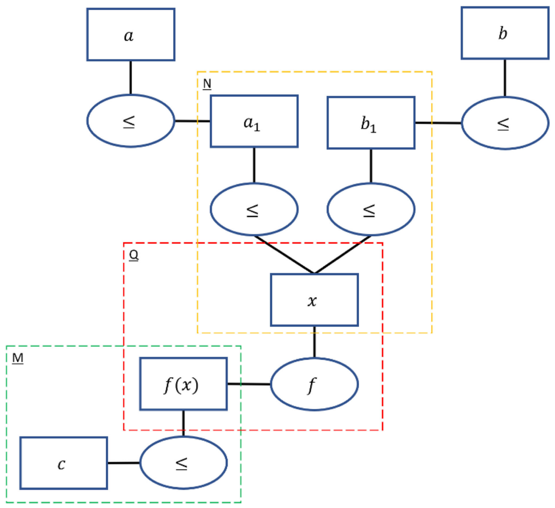

Fig. 3. Conceptual graph of a constraint model transformation.

a feasible region. A two- or higher dimensional "rectangle" formed by constraint relations on each variable is called an *orthotope* in multidimensional Euclidean space. If these are the only constraints on the design space, the design specification permits independent choices of values for each design variable (within the orthotope). Design decisions can be made "one factor at a time" because of the independence.

The feasible region though is usually constrained by further relations. Joint constraints on the objectives are transformed into further relations on the design variables that can change the geometry of the feasible region in ways that can complicate the specification of individual design variables. The transformed relations can arise from relations between design objectives and design variables, for example, a response surface $f : X \to Z$ that might be based on models from engineering or science. However, the (joint) specification of continuous design variables should always be in the form of an orthotope.

The CDD model transformation derives from the mapping $f^{-1} : P(Z) \to P(X)$ where $P(\cdot)$ denotes power set (i.e., the set of all subsets). For a 1-D response surface, a constraint given by $f(x) \leq c$ (for a given $x$ in the design space) implies a condition on the design specification: $a_i \leq x \leq b_i$ for each individual design factor. Fig. 3 is a conceptual graph of such a model transformation. CDD implements MDSD by specifying transformations like this. Fig. 4 depicts how this can affect the geometry of the design space for two key design variables in the emissions case study in Section V.

In general, every orthotope that satisfies the transformation of objective constraints into the feasible region is therefore a candidate solution (design specification) for the associated CDD problem. This family of orthotopes forms a relational structure on the constrained feasible region determined by the CDD transformation. This defines a *structural type* that can be called *constraint-driven structure (of the design space).*

The processes of interpreting concepts into design objectives followed by CDD model transformation into the design space are examples of technical processes for essential architecture definition, as given in Section IV-A.

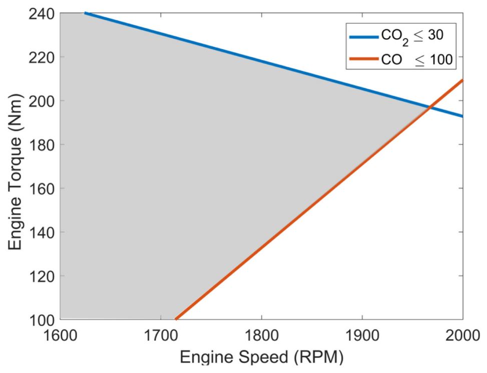

Fig. 4. ADAS architecture definition: Constrained design space.

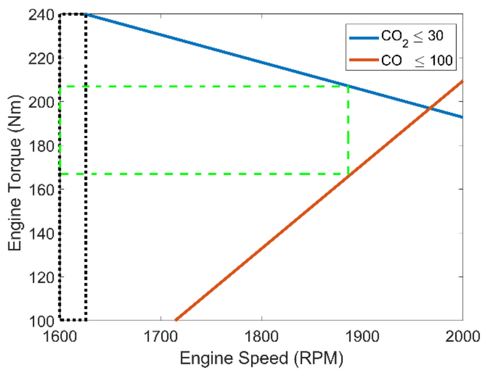

Fig. 5. Architecture of ADAS design space.

### D. Application to Advanced Driver Assistance System (ADAS) With Emission Control

A coordinated control architecture design [36] for a class of emissions control problems will now be investigated to demonstrate MDSD at the simplest possible level. A design space for an ADAS will be architected to integrate an emissions control function with an existing cruise control system. A modular system architecture will be assumed for which the integrated solution does not require redesign of the existing system. Essential architecture definition will be used to define an architecture for the design space.

Two design objective constraints will be specified for emissions and two design variables for the engine. In this simplified problem, the two objectives are $z_1 = \mathrm{CO_2}$ emission and $z_2 = \mathrm{CO}$ emission, measured in grams per kilometre. The constraints are $z_1 \le 30$ and $z_2 \le 100$ . The design variables will be engine speed, $x_1$ in revolutions per minute (r/min) and engine torque, $x_2$ in newton-meters (N·m). An engine map has been measured for the state space $1600 \le x_1 \le 2000$ and $100 \le x_2 \le 240$. As a mathematical mapping, the map is defined by two response surfaces $f(x_1, x_2) = z_1$ and $g(x_1, x_2) = z_2$. In practice, the map is normally a table of discrete measurements.

The outcome of CDD model transformation is the shaded region of the design space displayed in Fig. 4 by the transformation of the constrained objectives into engine torque and engine speed through the inverse mappings of the two response surfaces. The ADAS cruise-control functionality includes providing torque demands to the engine to maintain a constant vehicle speed (which in a fixed gear corresponds to a constant engine speed) and to accelerate the vehicle (which in a fixed gear corresponds to increasing the engine speed).

The integration strategy for the ADAS emissions control functionality is to mediate torque demanded by the driver before it is communicated to the existing cruise control system. The demanded torque presented to the system may need to be reduced in order to ensure that the emissions constraints are not violated.

At the start of either a constant speed cruise or an acceleration action, the engine is initially at a given state (r/min, torque) within the shaded region in Fig. 4. The simplest initial starting point would be toward the center of the region. When the demand for increased torque is sent to the engine, the outcome will be increased revolutions per minute. If the engine state starts from near the center, then there is no need to manage the relation between torque states and engine speed (for the purpose emissions control) until the state approaches one of the two boundaries. Keeping the state within a rectangular region (a 2-D orthotope) in the constrained design space is a simple and easy method to integrate ADAS emissions control functionality with the control system.

Any rectangle that fits within the shaded region is in principle suitable for specifying a region of control, i.e., a set of combinations of engine torque values and speeds that comply with the constrained design space. Rectangular regions such as these are orthotopes (2-D in this case). They are the basis for an interrelational structure on the constrained design space as well as design specification, i.e., a relational structure of orthotopes interrelated by shared relations with the constraints.

There are infinitely many orthotopes, two of which are depicted in Fig. 5. The "tall narrow" orthotope reflects allowing for maximum variation in torque but only in a narrow range of engine speeds. The restriction in engine speeds (which is seen by the narrowness of the orthotope) would be appropriate for the function of maintaining a constant vehicle speed (corresponding to engine speeds near 1600 r/min). The mediation of torque demands should be minimal in this case even over hills, as the existing cruise control system will manage the engine speed to maintain a constant vehicle speed. If the initial engine state is somewhat greater than 1600 r/min, a different orthotope will need to be selected toward the right of the one in the figure. This will necessarily reduce the vertical extent of the orthotope and consequently the range of torque demands permitted.

A more "balanced" orthotope is also shown in the figure, but it clearly reduces the range of torque outputs permitted. This orthotope could also be useful for the function of accelerating because the increase in engine speed for much of the region is not at risk of taking the engine state past the constraints on emissions.

Note that the two orthotopes in Fig. 5 enjoy the property of *maximality* within the constraint-driven structure. This means that neither of them can be contained by another orthotope that conforms to the constraints. In other words, an orthotope is maximal if any other orthotope that contains it must violate at least one of the constraints on the design space.

The result is an interrelational structure formed of "maximal" orthotopes. Identifying this set of orthotopes as a desirable (if not required) feature of candidate design solutions marks a completion of essential architecture definition in the MDSD process for this example. In engineering practice, specifying architecture for the design space can therefore help to point the designer as to where to find suitable solutions.

This design of ADAS illustrates how the process of essential architecture definition can guide the designer in the selection and specification of models based on structural types and architectural properties. The practical nature of the example also illustrates how the definition process for a system need not be a linear process. The designer is free to think in terms of models and later consider what types of structure they have been expressed in. For conceptual integrity and architectural concordance in design iterations though, structural types and architectural properties should be specified as early as possible.

Architecture definition as demonstrated in this example is seen as a general structuring process that provides a basis for modeling (among other things). The interplay between architecture definition and the specification of models is central to engineering practice, as defined in this section. The structural types that have been used are inherently simple but require insight, if not deep understanding of the system. Specifying the right types of structure is therefore critical to system design.

## V. Emissions Reduction Case Study

The case study will develop a high-level design for an emissions control system (ECS) that will demonstrate the engineering utility of the definitions and processes offered in this article. The design will feature but be limited to a high-level object-oriented PIM level specification for the ECS software and a calibration method that can be used to reduce emissions. The object-oriented models for the MDA PIM will be linked to engineering models for engine calibration using conceptual structures as in Section IV. As previously noted, this provides conceptual integrity and architectural concordance (in the ECS). The case study will then introduce recent advancements in commercial quality calibration methods [37] by building on the design space architecture for the ADAS in Section IV. Algorithms can be implemented using a relational-oriented mathematical framework that can accommodate structures associated with maximal orthotopes in the design space. The case study concludes with a consideration of how the definitions and processes also can be part of systems engineering standards.

### A. *Engine Calibration Problem Definition*

In the case study, the problem to be solved is that emissions from automotive diesel engines must not exceed regulatory constraints when the vehicle is driven in a variety of profiles, such as those specified in [38]. The primary source of emissions originates from inefficient combustion in the engine that leads to undesired chemical products, such as nitrogen oxides ($NO_X$). In modern diesel engine cars, emissions are often reduced using aftertreatment technologies, such as diesel particulate filters and lean $NO_X$-traps.

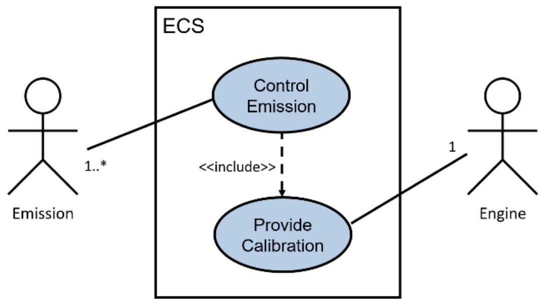

Fig. 6. Use case diagram for the ECS.

One approach to improve the efficiency of the chemical reactions that occur within the engine combustion chambers is to design an optimal engine map. In order to achieve current regulatory constraints, a lower-level engine map is used that involves tuning of the engine calibration parameters, such as mass air flow (MAF), fuel rail pressure (FRP), and low pressure exhaust gas recirculation (LP-EGR).

The ECS controls emissions (indirectly) through the provision of calibration data to the engine. This is the basis of the functionality of the ECS. When the ECS is considered as a system in this case study, it will have direct interactions with the engine and an ADAS that mediates commands from the driver to the engine, and which informs the ECS of the commands sent. The system elements will be an electronic control unit with associated software applications and an engine map. The functionality and behavior of the ECS will be the subject of Section V-B. The association of calibration methods for managing the engine map will be the subject of Section V-C.

### B. *Specification of High-Level MDA PIM*

MDA development focuses first on the functionality and behavior of a system by means of a PIM that separates application logic from underlying platform technology. A use case diagram will be specified for system functionality and an activity diagram for system behavior. For simplicity, different levels of abstraction will be used: the ADAS will be included in the activity diagram but not the use case diagram. Also, the emissions will not be included in the activity diagram (The diagrams are an output of architecture definition as discussed earlier.). Software implementation and PSM level details of this elementary software architecture for the ECS will not be pursued. These are routine in MDA. The modeling implications for system architecture in the case study are of greater interest than the detailed modeling of software.

Each of the two use cases in Fig. 6 depicts something that the ECS system is intended to do: control emissions (from the engine) and provide calibrations (to the engine). Each is an "interaction" relation between the system and an "actor" (in this example, the engine or its emissions). An inclusion relation between the two use cases is used to model the indirect effect of the ECS on emissions. As in Section IV, the use cases depict a relational structure comprising two relations that are an interpretation of system functionality.

As in Section IV-A, two types of graphical models can be constructed that interpret the sentences associated with the

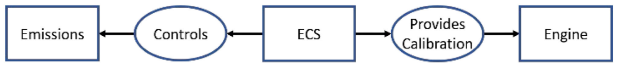

Fig. 7. Conceptual graph for ECS functionality.

functional requirements. The result is depicted in Figs. 6 and 7. The conceptual graph in Fig. 7 (which is comprised of two relations joined by the ECS) also represents a relational structure comprising the two relations that are an interpretation of system functionality. This view of system functionality is external to the ECS (i.e., interactions with its environment). Internal details of the system are suppressed.

However, when the two figures are closely compared, it is apparent that Fig. 6 has a relation of *inclusion* between the two use cases, which cannot be captured as a relation in the conceptual graph in Fig. 7. The syntax of a conceptual graph must conform to the predicate calculus, which only permits relations between entities (e.g., concepts) and not relations between relations. Inclusion is an example of interrelationship.

The *functional structure* of the ECS is therefore represented in Fig. 6 by two relations (use cases) and one interrelation (inclusion). Note then that interaction is sufficient to define *system functionality* but not sufficient to define the *functional structure* of the system (when the internal details are considered).

The structure represented in Fig. 6 is an example of an object-oriented structure that can be used to model the interrelations of the *system*. The interrelational structure creates an association between *system* and *architecture* (as noted in Section I). Systems endowed with a structure of this class can be said to conform to the functional architecture of the ECS.

It should also be noted that the use case relations depicted in the conceptual graph (see Fig. 7) and the use case diagram (see Fig. 6) are semantic predicates. As depicted, the statements in the figures are undecidable. They are sentences that have partially interpreted models in the sense of Tarski theory. There is not yet enough information to determine if the predicate sentence (e.g., a well-formed formula quantified by an existential operator) is true or false (*in the domain*). The engineering problem is to specify the system (model) at a level of detail from which it can be realized in conformance to the functional architecture as well as other defined requirements. This will validate the fully interpreted model and is concept realization.

An activity diagram must also be given to complete the specification of the PIM. The influence of the ECS on the emissions from the engine is achieved by its use of the engine map and a provision of calibrations in real time to the engine.

The diagram in Fig. 8 represents the high-level behavior of the ECS in its relation to the ADAS and engine. It depicts a structure that has been realized through structures of behavioral type. This type of structure could also be represented in a conceptual graph, but this will not be needed in this case study.

Use case and activity diagrams represent the functionality and behavior in the PIM for the software architecture of the ECS. Specifying a calibration method will complete the design.

## C. Structures for Calibration Method Design

In this section and the following section, an integration of the ROSETTA framework and axiomatic design [39] will be used

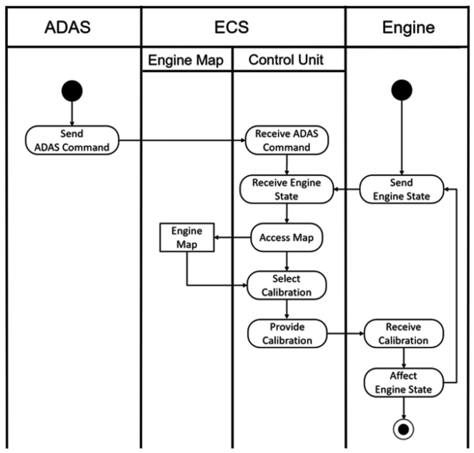

Fig. 8. ECS activity diagram.

to show how a calibration method can be developed to seek robust calibrations for emissions control. In a modern automotive vehicle, dozens of calibration factors must be considered, but in practice, a dozen or fewer dominant factors are typically sought. These will become the design variables in an MDSD problem. As with the ADAS example, maximal orthotopes for these variables will be identified within a constrained design space. For simplification, the problem will be reduced to three design objectives and three design variables that are joined together by three response surfaces.

A matrix formulation of ROSETTA provides a mathematical implementation of MDSD that is a framework similar to the quality function deployment house of quality [40], but can be used to translate expert opinion into mathematical relations. For this reason, the name of the framework bears an intentional similarity to the Rosetta Stone that provided the means to interpret between the Greek, Egyptian, and hieroglyphics demotic languages. In an essential architecture definition process for CDD, ROSETTA provides a facility for interpretation of models between systems engineering and systems (and software) architecting. It supports rigorous development of structures for system design, as in the example in Section IV-D. ROSETTA will be used to extend constrained design spaces, such as in Fig. 4, to higher dimensional spaces.

A mathematical implementation of MDSD using ROSETTA is depicted in Fig. 9. The dots are normalized data from a design of experiments. The data are structured using binary relations depicted in the $1 \times 1$ matrices that correspond to pairings of the design objectives and design variables (These matrices represent a structural type of binary relationship.), Outcomes of model transformations, such as the one in the conceptual graph of Fig. 3, are depicted as regions bounded by the dashed lines (that correspond to emission levels constrained by regulations). The feasible design points are those that comply with the constraints. Fig. 9 is a multivariate expression of Fig. 3.

The aggregation of the pairings of the three design objectives (FC, $NO_x$, and PM) with each other form a $3 \times 3$ structure, which is referred to as the $\underline{M}$-matrix. The cells contain the constrained binary relations of the requirements. This is the objectives matrix. Similarly, the design matrix is referred to as $\underline{N}$ and is formed by the aggregation of the pairings of the

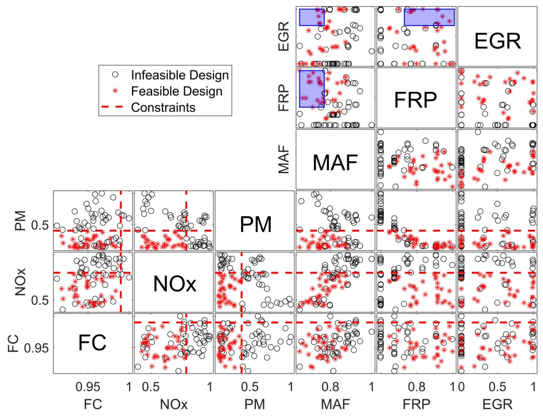

Fig. 9. ROSETTA framework for emissions control.

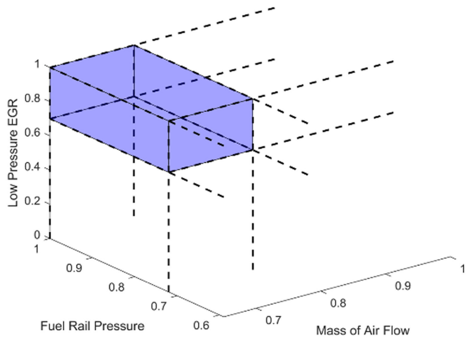

Fig. 10. Maximal orthotope for emissions control.

design variables (MAF, FRP, and EGR). The Q-matrix depicts the relationships between the design objectives and the design variables. It corresponds to a transformation of the design objectives into the design variables that will further constrain design solutions. The feasible designs are thus transformed into the N-matrix but the simplicity of the constraint lines in the M-matrix is lost. The 2-D projections of the maximal orthotope in Fig. 10 into the ROSETTA framework (i.e., the shaded rectangles in the cells) make it possible to visualize higher dimensional solutions.

Relations in the Q-matrix can be mathematically approximated by a set of response surfaces given by $z_i = f_i(x_1, x_2, x_3)$. Each matrix element in the Q-matrix also corresponds to the sensitivity between a pair $z_i$ and $x_j$. The gradient line slopes are calculated by taking the partial derivative $(\partial z_i)/(\partial x_j)$ of the response surface at the initial design point. In this specific example, the response surfaces are approximated by pure-quadratic functions of the form

$$z_i = \beta_0 + \sum_{j=1}^{N} \beta_j x_j + \sum_{j=1}^{N} \beta_{jj}(x_j)^2. \tag{3}$$

The sets of normalized coefficients $\beta$, as presented in Table I, have been derived from an engine test bed for an engine speed of 1350 r/min and a torque demand of 25 N·m. As such, the slopes of the local gradients are determined by the expression

$$\left.\frac{\partial z_i}{\partial x_j}\right|_{x_j = x_{j0}} = \beta_j + 2\beta_{jj}x_{j0} \tag{4}$$

TABLE I
RESPONSE SURFACE COEFFICIENTS

| | $\beta_0$ | $\beta_1$ | $B_2$ | $B_3$ | $\beta_{11}$ | $B_{22}$ | $B_{33}$ |
|---|---|---|---|---|---|---|---|
| $CO_2$ | 5.97 | -1.21 | -11.31 | -0.07 | 0.30 | 6.27 | 0.03 |
| NOx | -4.01 | 6.53 | 2.89 | -0.24 | -2.37 | -1.72 | 0.03 |
| Soot | 1.22 | -0.34 | -0.42 | -0.02 | 0.27 | 0.27 | 0.03 |

where the initial design value for $x_j$ is denoted by $x_{j0}$.

In higher dimensional multiobjective multiattribute problems, the approximation by response surfaces in the matrix representation provides a method to reduce complexity in the system analysis. Therefore, ROSETTA can be used both to structure information and to manage complexity.

## D. *Specification of a Calibration Method*

The use of maximal orthotopes, as in Fig. 5, has been extended to the emissions problem. ROSETTA and the Nam Suh axiomatic design theory have been integrated to develop algorithms to seek maximal orthotopes of calibrations derived from an engine map to be provided to the engine in real time, as in the activity diagram depicted in Fig. 8.

The result is a recent advancement in commercial quality calibration methods that has been published in a patent with Jaguar Land Rover [37] to which the reader is referred for details of the algorithm. The essential idea is to rank the variables then create a maximal orthotope one factor at a time by pushing each factor to its constraint limit in the order of the ranking. Because of the interrelationships between the design variables (i.e., the calibration factors), every step of the process reduces the available design space. Fig. 10 depicts a maximal orthotope associated with the framework in Fig. 9.

## E. *MBSE Standard*

The concepts of CDD are therefore seen to have substantive engineering utility in the context of MBSE based on relational structures formed of constraints on design objectives that are transformed into structures in the design space. The engineering utility of the essential definitions and the architecture definition process offered in this article have been clearly demonstrated. Standardization of terms and concepts in MBSE can also benefit from the definitions, architecture definition process, and mathematical interpretations put forth in this article.

A standard that provides evidence of this claim is the OMG UPR (the UML profile for ROSETTA) [35]. The profile specifies a comprehensive facility to structure information in support of model-based analysis for architecture optimization and system design. The facility can also support efficient and effective transformation of the system architecture into detailed system design. Such a facility could be used, for example, to capture and structure information from a system model for a complex system, such as that of the calibration problem for emissions reduction.

UPR supports information structuring for the *quantification* and *model transformation* portions of MDSD depicted in Fig. 2 but was not intended to define new techniques for model-based analysis and optimization. The implementation of CDD through

ROSETTA, on the other hand, is intended as a new technique. Information captured through a UPR facility can be used to populate the framework in a way that maintains conceptual integrity of the various models of interest.

UPR comprises five normative packages: foundations, operators, design objective constraints, design variable constraints, and relational structure. Operator and constraint stereotypes provide interpretations into UML in each of the binary relations in Fig. 3 for CDD model transformation. The mapping $f(x)$ can also be interpreted when it is in the form of a polynomial response surface, as in Fig. 9 and Table I.

Therefore, conceptual graphs as in Fig. 3, matrix arrays as in Fig. 9, and the UPR diagrams (that implement its stereotypes) can be used to implement models in structures of constraint driven type, as introduced in Section IV-C. The essential definitions for structure and architecture together with the explanation of Tarski model theory and conceptual structures in Section III offer a mathematical basis for UPR.

## VI. Conclusion and Future Work

New intuitive fundamental definitions complemented by mathematical interpretation have been contributed and used to specify a mathematically based technical process for architecture definition that can be applied to the life-cycle specification of a system. Essential architecture definition is a general structuring process that involves: defining theories, specifying symbols for the language associated with the theories, and specifying fully interpreted models of the theories. Its validity has been demonstrated through several elementary examples in the practice of system and software design. Engineering utility has been demonstrated in a detailed case study on recent advancements in commercial quality calibration methods for diesel engine emissions and by effective modeling of CDD problems for which the OMG has recently adopted an open international standard (UPR: the UML Profile for ROSETTA).

Ambiguities in the adopted definition of architecture have been clarified through mathematical interpretation. As defined in this article, *architecture* is consistent with the adopted definition but is at a higher level of abstraction. Models are interpretations of concepts (theories) into structures. Architecture specifies structural type and properties to be implemented.

The essential definitions, the essential architecture definition process, and advances put forward in this article will be carried forward along lines of both research and commercialization. The advances are being applied to a facility and methods for CDD and to the electronic architecture of vehicles [41]. The research will use category theory as an advanced structuring method for architecture definition. New formalisms can enhance systems engineering tool development and evolve the scientific basis of the discipline.

## References

[1] M. K. Wilkinson, C. E. Dickerson, and S. Ji, "Concepts of architecture, structure and system," Dec. 2018, *arXiv:1810.12265v2*.

[2] L. Von Bertalanffy, *General Systems Theory*. New York, NY, USA: Braziller, 1969.

[3] A. W. Wymore, *Model-Based Systems Engineering*. Boca Raton, FL, USA: CRC Press, 1993.

[4] R. Rosen, "On models and modeling," *Appl. Math. Comput.*, vol. 56, no. 2/3, pp. 359–372, 1993.

[5] C. E. Dickerson, S. M. Soules, M. R. Sabins, and P. H. Charles, *Using Architectures for Research, Development, and Acquisition*. Washington, DC, USA: Office Assistant Secretary Navy, 2004.

[6] C. E. Dickerson, "Towards a logical and scientific foundation for system concepts, principles, and terminology," in *Proc. IEEE Int. Conf. Syst. Syst. Eng.*, 2008, pp. 1–6.

[7] M. K. Wilkinson, *An Architect's Manifesto for the Development of Systems Engineering*. Ilminster, U.K.: INCOSE UK, 2016.

[8] F. P. Brooks, Jr., *The Mythical Man-Month*. Reading, MA, USA: Addison-Wesley, 1995.

[9] C. E. Dickerson and D. N. Mavris, *Architecture and Principles of Systems Engineering*. Boca Raton, FL, USA: CRC Press, 2010.

[10] *Systems and Software Engineering—Architecture Description*, ISO/IEC/IEEE 42010:2011, 2011.

[11] A. Tarski, "Contributions to the theory of models. I," *Indagationes Mathematicae*, vol. 16, pp. 572–581, 1954.

[12] A. Tarski, "Contributions to the theory of models. III," *Indagationes Mathematicae*, vol. 17, pp. 56–64, 1955.

[13] S. Eilenberg and S. Mac Lane, "General theory of natural equivalences," *Trans. Amer. Math. Soc.*, vol. 58, pp. 231–294, 1945.

[14] System and Software Engineering —System Life Cycle Process, ISO/IEC/IEEE 15288:2015, 2015.

[15] M. R. Emes *et al.*, "Interpreting 'systems architecting'," *Syst. Eng.*, vol. 15, pp. 369–395, 2012.

[16] Software, Systems and Enterprise—Architecture Processes, ISO/IEC/IEEE 42020:2019, 2019.

[17] *Software, Systems and Enterprise—Architecture Evaluation Framework*, ISO/IEC/IEEE 42030:2019, 2019.

[18] R. F. Hilliard, T. B. Rice, and S. C. Schwarm, "The architectural metaphor as a foundation for systems engineering," in *Proc. INCOSE Int. Symp.*, 1996, vol. 6, no. 1, pp. 559–564.

[19] *IEEE recommended practice for architectural description for software-intensive systems*, IEEE 1471-2000, 2000.

[20] C. S. Wasson, *System Analysis, Design, and Development: Concepts, Principles, and Practices*. Hoboken, NJ, USA: Wiley, 2006.

[21] R. Cloutier, "System definition," SEBoK, 2019. [Online]. Available: *https://sebokwiki.org/w/index.php?title*=System_Definition&oldid=56369

[22] "Model Driven Architecture (MDA): The MDA Guide Rev 2.0," Object Management Group, 2014. [Online]. Available: https://www.omg.org/mda/presentations

[23] S. J. Kapurch, *NASA Systems Engineering Handbook*. Washington, DC, USA: NASA, SP-2016-6105 Rev 2, 2016, p. 44.

[24] *Oxford English Dictionary*. London, U.K.: Oxford Univ. Press, 2019.

[25] J. L. Bell and A. B. Slomson, *Models and Ultraproducts*. Amsterdam, The Netherlands: North Holland, 1969.

[26] W. Hodges, "Model theory," Stanford Encyclopedia of Philosophy, 2013. [Online]. Available: https://plato.stanford.edu/entries/model-theory/

[27] J. F. Sowa, *Conceptual Structures: Information Processing in Mind and Machine*. Reading, MA, U.S: Addison-Wesley, 1983.

[28] Information *Techonology*—Common Logic (CL): A *Framework* for a *Family* of *Logic-Based Languages*, ISO/IEC 24707:2018, 2018.

[29] C. K. Ogden and I. A. Richards, *The Meaning of Meaning*. London, U.K.: Routledge & Kegan Paul, 1923.

[30] J. F. Sowa, *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Pacific Grove, CA, U.S: Brooks/Cole, 2000.

[31] C. E. Dickerson, "A relational oriented approach to system of systems assessment of alternatives for data link interoperability," *IEEE Syst. J.*, vol. 7, no. 4, pp. 549–560, Dec. 2013.

[32] J. K. Finch, *Engineering and Western Civilization*. New York, NY, USA: McGraw-Hill, 1951

[33] *IEEE Standard for Application and Management of the Systems Engineering Process*, IEEE 1220-2005, 2005.

[34] R. Cloutier and M. Bone, "MBSE survey presented January 2015 INCOSE IW los angeles," OMGWiki, Jan. 2015. [Online]. Available: http://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:incose_mbse_survey_results_initial_report_2015_01_24.pdf

[35] UPR: UML Profile for ROSETTA, Object Management Group, Jun. 2018. [Online]. Available: https://www.omg.org/spec/UPR/About-UPR/

[36] T. C. Lin, S. Ji, C. E. Dickerson, and D. Battersby, "Coordinated control architecture for motion management in ADAS systems," *IEEE/CAA J. Automatica Sinica*, vol. 5, no. 2, pp. 432–444, Mar. 2018.

[37] C. E. Dickerson, S. Ji, and D. Battersby, "Calibration system and method," U.K. Patent GB2555617, May 9, 2018. Online: https://www.ipo.gov.uk/pipsum/Case/PublicationNumber/GB2555617

[38] T. J. Barlow, S. Latham, I. S. McCrae, and P. G. Boulter, "A reference book of driving cycles for use in the measurement of road vehicle emissions," Transport Res. Lab., Crowthorne, U.K., Published Project Rep. PPR354, 2009.

[39] C. E. Dickerson and D. Mavris, "A brief history of models and model based systems engineering and the case for relational orientation," *IEEE Syst. J.*, vol. 7, no. 4, pp. 581–592, Dec. 2013.

[40] D. Mavris, K. Griendling, and C. E. Dickerson, "Relational-oriented systems engineering and technology tradeoff analysis framework," *Amer. Inst. Aeronaut. Astronaut. J. Aircr.*, vol. 50, pp. 1564–1575, Aug. 2013.

[41] C. Dickerson and S. Ji, "Analysis of the vehicle as a complex system, EPSRC," *Impact*, vol. 2018, no. 1, pp. 42–44, 2018.

[42] A. Tarski, "Contributions to the theory of models. II," *Indagationes Mathematicae*, vol. 16, pp. 582–588, 1954.

**Charles E. Dickerson** (Senior Member, IEEE) received the Ph.D. degree in mathematics from Purdue University, West Lafayette, IN, USA, in 1980.

He is currently a Professor and Chair of systems engineering with Loughborough University, Loughborough, U.K. He was a Technical Fellow with BAE Systems, Arlington, VA, USA. His aerospace experience further include the Lockheed Skunk Works and Northrop Advanced Systems. He was also a member of the Research Staff with Lincoln Laboratory, Massachusetts Institute of Technology. He has served secondments as the Aegis Systems Engineer for US Navy Ballistic Missile Defense and the Director of Architecture for the Navy's Chief Engineer. He is currently Co-Chair of mathematical formalisms in the Object Management Group.

**Michael Wilkinson** received the B.Sc. degree in physics and the Ph.D. degree in theoretical physics in 1985 from King's College, London, U.K.

He is currently a Chief Technologist with BAE Systems Maritime, Barrow-in-Furness, U.K., and is a Visiting Professor with Loughborough University, Loughborough, U.K. He was a Technical Director and Professional Head of discipline for systems engineering with Atkins. Prior to that, he was a Technical Director of the Niteworks partnership of the MOD. He has served as the President and Academic Director of the UK Chapter of the International Council on Systems Engineering. He is currently Co-Chair of the UK Chapter AWG.

**Eugenie Hunsicker** received the B.A. degree from Haverford College, Haverford, PA, USA, in 1992, and the Ph.D. degree from the University of Chicago, Chicago, IL, USA, in 1999 both in mathematics.

She is currently a Senior Lecturer in mathematics with Loughborough University, Loughborough, U.K. She was with Lawrence University, Appleton, WI, USA. Her research interests include analysis and topology, as well as applied statistics, especially for high-dimensional and non-Euclidean data.

Dr. Hunsicker is a member of the London Mathematical Society and the Royal Statistical Society.

**Siyuan Ji** (Member, IEEE) received the M.Sc. degree and the Ph.D. degrees in 2015 from the University of Nottingham, Nottingham, U.K. both in physics.

He is currently a Lecturer with the Department of Computer Science, University of York, York, U.K. He was a Research Associate with Loughborough University. His research interests include systems engineering methodologies, model-based system safety and reliability assessments, product line engineering, and constraint-driven design algorithms.

**Mole Li** (Member, IEEE) received the bachelor's degree in computer science from the University of Hull, Hull, U.K., in 2012, and the M.Sc. degree in 2013 from Loughborough University, Loughborough, U.K., where he is currently working toward the Ph.D. degree.

He is currently a Technical Specialist in Rolls-Royce Control Systems, Derby, U.K., supporting model-based systems engineering (MBSE) standardization and developing methodologies and tools for whole engine life-cycle development. His research interests include integration of software product line engineering with MBSE.

**Yves Bernard** received the master's degree in biological oceanography from Pierre et Marie Curie Paris VI University, Paris, France, in 1989.

He is supporting model-based systems engineering for the Process, Method, and Tools Department, Airbus Defence and Space, Toulouse, France. He is with the Airbus Avionics Department, where he is responsible for development of model-based methods applicable to critical embedded systems, including both software and hardware components. He has been involved in a number of OMG standards to include UML 2.4 - 2.6, Precise Semantics of UML Composite Structures and State Machines (PCSC 1.0 and PSSM 1.0), Modeling and Analysis of Real Time Embedded Systems (MARTE 1.1), and UPR 1.0. He is the Co-Chair of the Revision Task Force for the systems modeling language (SysML) and a member of the SysML 2.0 team.

**Graham Bleakley** received the B.Eng. degree in mechanical engineering from Southbank University, London, U.K., in 1992, and the Ph.D. degree in model-based systems engineering and design from the City University, London, U.K., in 2001.

He is currently a Co-Chair and Lead Architect with the OMG Unified Architecture Framework, which unifies DoDAF, MODAF, and NAF into a single metamodel and commercially available framework for enterprise architecture and systems of systems development. He is also the IBM representative for SysML 2.0 at OMG.

**Peter Denno** received the B.A. degree in mathematics from the University of Connecticut, Storrs, CT, USA, and the master's degree in supply chain management from Pennsylvania State University, State College, PA, USA. He is currently working toward the Ph.D. degree in systems and industrial engineering with Loughborough University, Loughborough, U.K.

He is currently a Computer Scientist with the Engineering Laboratory, National Institute of Standards and Technology, Gaithersburg, MD, USA, where he has led projects to facilitate the use of analytical methods in manufacturing. He was in manufacturing and systems engineering with Pratt & Whitney.