Received Date: 7/23/2012, Revised Date: 3/4/2013

A Brief History of Models and Model Based Systems Engineering and the Case for Relational Orientation

Charles E. Dickerson, *Member*, IEEE

Dimitri Mavris, *Member*, IEEE

Abstract – Models are at the heart of science and engineering. Model based approaches to software development and systems engineering use technologies to include graphical modeling languages such as the Systems Modeling Language (SysML) that support system design and analysis through machine readable models. This paper traces key historical contributions of software and systems engineers over the past five decades beginning with Yourdon and Wymore to show a coherent concept of models and how they can be used for software and systems engineering. Recent model based systems engineering (MBSE) methodologies supported by commercially available modeling tools are also summarized. Relational Orientation is seen to be the underlying viewpoint that expresses and binds these approaches. Relational Orientation for Systems Engineering (ROSE) is then specified using a general systems methodology. Systems are seen to access each other's models in ROSE much like classes in Object Orientation access each other's objects. Object oriented frames for software engineering are extended to relational frames to specify an innovative framework for system design and analysis. This generalizes the axiomatic design approach of N.P. Suh. A repeatable procedure supporting greater concurrency between design and verification is also demonstrated for searching the solution space in linear axiomatic design.

***Index Terms*: Model, graphical modeling languages, model based systems engineering, software engineering, model driven architecture, first order models, relational structures, homomorphism, linear optimization, relational frames, OOSEM.**

I. Introduction

MODEL BASED APPROACHES to systems are well understood and practiced in science and mathematics based on the foundation provided by mathematical logic. In order to make models more visual and intuitive, software and systems engineers have developed various graphical modeling languages. The result has been the development of machine readable languages by the Object Management Group (OMG), which are commercially supported and provide a critical technology for model based software and systems approaches.

A. *Models in Science and Mathematics*

A model of a system is generally regarded as a representation of the system. Models are also abstractions that suppress details not of interest. In science, mathematical models of a system bring precision that can be used to predict properties and behaviors of systems. Hawking, for example in [1], attributes a good model in physics to be:

- Simple
- Mathematically correct
- Experimentally verifiable

In this viewpoint, mathematics becomes the modeling language of physics.

The language of mathematics is the Predicate Calculus of logic. The term *model* in mathematical logic has a specific and formal meaning. Specifically, a model of a sentence is a relational structure for which an interpretation of the sentence expressed in the Predicate Calculus is valid (true) within that structure. A *relational structure* is a collection of mathematical relations on a defined set. A *mathematical relation* is a collection of relationships between the elements of the set, e.g. $R_O = \{(m, n) \mid m < n\}$ is an ordering relation on the system of counting numbers and each pair (m, n) is a relationship. Thus, *relationships* are instances of relations. *Interpretations* in mathematical logic are one-to-one mappings of sentences in Predicate Calculus into relational structures.

This concept of a model in logic is made clear by the example of interpretation of the sentences in an axiom system for an elementary geometry. An *axiom system* is comprised of a collection of key words that remain undefined (to avoid circularity) and sentences, called axioms that establish relations between the key words. An axiom system for an elementary geometry might include the terms *point* and *line* as key undefined words; and the axioms for the geometry might include the following three sentences about the key words:

(i) Every *line* is comprised of at least two *points*.

(ii) There exists a *line* passing through each *point*.

(iii) Every *pair of lines* intersects in at most one *point*.

The meaning of the key words is determined by: (a) relations between the words established by the axioms and (b) interpretation of the axioms into specific models (relational structures). This suggests that there are two types of models needed for every system. One type is for the intrinsic meaning of the sentences (axioms) expressed as relations between the key words. The other type is for interpretation of the key words that contributes new meaning to the sentences. Fig. 1 illustrates two interpretations of the elementary geometry

Manuscript received 29 December 2011; revised 4 March 2013. This work was supported in part by the Royal Academy of Engineering and the Engineering and Physical Sciences Research Council, UK.

C.E. Dickerson is with Loughborough University, Leicestershire, LE11 3TU, UK

D. Mavris is with the Georgia Institute of Technology, Atlanta, GA 30332 USA

Digital Object Identifier….

axiom system. The first is a planar geometry in which the interpretation of all three the sentences is true. This then is a valid model of the axiom system. The second interpretation is a spherical geometry in which the interpretation of a *line* is that of a great circle on a sphere. In this case, the third sentence is false because great circles intersect at two points. The interpretation as a spherical geometry is therefore not a valid model of the axiom system.

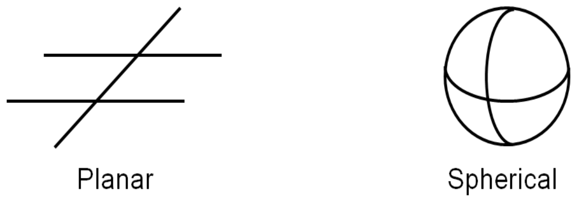

Fig. 1. Two Interpretations of an Elementary Geometry System

B. *Comparison with System Specification*

The concept of a model offered by mathematical logic and this simple example can give significant insight into system description and specification, which are subject to the same rule of logic that geometry is. Typically a system specification is a collection of sentences, much like an axiom system. A design then becomes a model of the system that is an interpretation of the system specification. The model should be simple, mathematically correct and experimentally verifiable. However, as with the geometry system, it should be expected that there can be multiple interpretations of a given specification, some of which may not be valid. Thus, some interpretations of the specification will be valid models of the system and others will not. The precision with which the interpretations are made is the subject of transformation between models which is first introduced in Section IV.

C. *Graphical Modeling Languages for Systems*

Graphical modeling languages for software development have been used for system description and specification from the early years of computer systems. These types of language are useful for visualizing concepts. Graphical models generally represent the entities of a system as nodes in a graph and relationships as arcs. The sentences carry the key words and the semantics of the association between the key word. The syntax and semantics provided by graphical models helps to capture the meaning of natural language sentences. However to capture the full meaning of the sentences requires interpretation of the graphical models into a machine readable language such as XML/XMI. Three graphical languages are reviewed: Entity-Relationship (E-R) Diagrams, the Unified Modeling Language (UML), and the Systems Modeling Language (SysML). E-R Diagrams were an early approach to data modeling. The UML specifies a language for object modeling and software development. SysML is a general-purpose graphical modeling language for specifying, analyzing, designing, and verifying systems. Open specifications for UML and SysML have been adopted by the Object Management Group (OMG).

1) Entity-Relationship Diagrams: The E-R Diagram is a data modeling approach introduced by Peter Chen in 1976 [2]. E-R is a high level data modeling notation that integrates the concepts of semantic modeling and Object Oriented modeling. Semantic modeling is used by linguists to represent the meaning of words and by artificial intelligence researchers for knowledge representation. The core concepts and terminology of E-R have much in common with those of UML. This is to be expected, as E-R, UML, and other graphical modeling approaches have their roots in mathematical logic.

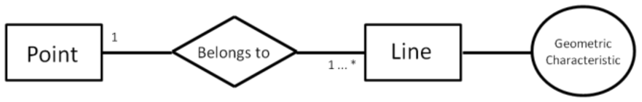

Fig. 2 Basic Elements of E-R Notation

In the graphical notation of E-R, the three basic elements of a diagram are: *entities* depicted by rectangles, *relations* represented by diamonds, and *attributes* depicted by circles. In each case, the name of the entity, attribute, or relationship is annotated inside the node. The nodes are connected by lines, which can be regarded as associations between the three types of elements. Fig. 2 depicts the E-R notation applied to the second axiom of the Elementary Geometry System. The relation depicted in the figure is a binary relation, i.e. two entities are associated with each other by the relation.

Multiplicities can also be included as in Fig. 2: '1' is associated with the entity Point and '1 … *' is associated with Line. The multiplicity specifies how many instances of an entity participate in an instance of the relation. Each instance of a relation is one of the relationships that comprise the relation. Note that this usage of the terms relation, relationship, and instance is precisely the same as the mathematical definition given in Subsection A.

2) Unified Modeling Language (UML): This language is the de facto standard for software development. It is a graphical language that provides semantics and notation for object-oriented problem solving. Models are important to software development for both engineering and communication, just like how blueprints drawn by architects are used in the construction of buildings. The more complicated the building, the more critical the communication between architect and builder, and the architect and the customer. There are many excellent references on UML and the OMG has an open website for tutorials: http://www.uml.org/#Links-Tutorials.

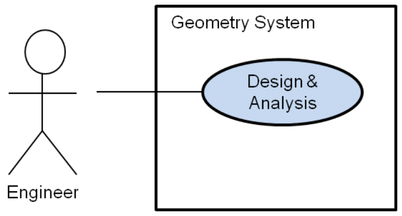

Fig. 3 Basic Elements of UML Use Case Diagram

The basic artifacts of UML can be used in systems engineering. These include the Use Case Diagram, Class Diagram, Package Diagram, Sequence Diagram, and State

Transition Diagram. Holt [3], among others, offers an overview of using UML in systems engineering organized around this core body of artifacts.

The Use Case Diagram is an external view of a system in which the interactions of actors with the system represent functional requirements graphically as illustrated in Fig. 3. The graphical elements of the diagram are the actor(s) (typically represented by named stick figures), the system (a box with the system name inside), the interaction(s) (denoted by communication line(s) between the actor and system), and the function(s) the system is used for by the actor (denoted by named oval(s) inside the box). Note that an interaction is a type of relation. The external viewpoint of the diagram also makes it part of the definition of the system boundary (as denoted by the box), i.e. the actors are identified as entities associated with the system that do not belong to it.

UML Classes are abstractions of the entities of the system. In software development, UML Classes become Objects when instantiated. A class has attributes and methods, which are called out within the diagram. Methods in the notation are distinguished from attributes by the parenthesis that follows the name of the method. Class Diagrams model relations between classes with a notation similar to the E-R Diagram.

Fig. 4 represents the relations described by the sentences in Axioms (i) – (iii) using three UML Class Diagrams. The diamond symbol in the diagram for Axiom (i) is 'aggregation', and indicates that class Line is comprised of the class Point. When these three graphical models are instantiated over a set (such as the Cartesian plane) they become a structure that interprets the relations described in the three axioms.

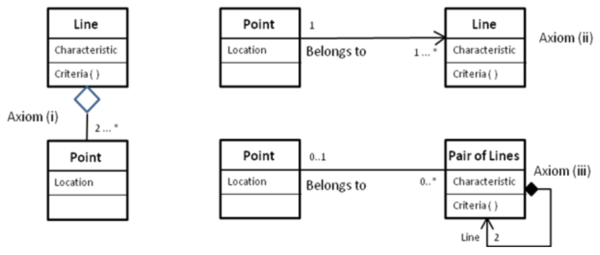

Fig. 4 UML Class Diagrams for Axiom System

While the UML Class Diagram has a similar content and form as E-R, there are minor differences. These are easily seen in the Axiom (ii) Class Diagram depicted in Fig.4, which is the UML equivalent of the E-R diagram in Fig. 2. In the Class Diagram, relations are the lines connecting the classes and text that defines the relation is placed directly on the line. The corresponding diagram in E-R has two lines; one for each of the entities (classes) associated with the relation (E-R symbol diamond), and the text identifying the relation is found inside the symbol. Also, the reference to the attribute in the Class Diagram is done within the Class notation whereas E-R uses a separate symbol (the circle).

States in physical systems are the values taken on by the system attributes over time. However, UML classes also have methods (operations executed and services accessed) in addition to attributes. Therefore 'state values' in Object Oriented modeling must also include the 'value' of the method, i.e. is it idle or active.

A system model represented by classes can be organized using UML Packages, which gather uniquely named model elements and diagrams into groups. A UML Package provides an encapsulated name space. Packages and groups of packages become system components in the model. The UML Package Diagram exhibits the individual packages and their dependencies as client-server relations. Modeling system components and dependencies this way provides a model of the system structure which can be used as part of the specification of the system architecture.

Components can be organized by specifying architectural domains, which are groups of packages defined by a common property, affinity, or governance. In both software and systems engineering, architectural domains should relate to system components and their organization. In model driven software development, e.g. in [4], the system architecture can be organized around four domains (groups of packages): the application specification, services accessed, the software architecture, and the implementation specification.

3) Systems Modeling Language (SysML): This is a graphical modeling language that extends UML for use in model based systems engineering (MBSE). It provides semantics and notation for systems engineering to support the specification, analysis, design, verification and validation of systems that include hardware, software, data, parametrics, personnel, procedures, and facilities. SysML supports interchange of models and data via XMI and AP233. It has been implemented in tools provided by a variety of vendors. Friedenthal [5] provides a good introduction to SysML and its use in MBSE.

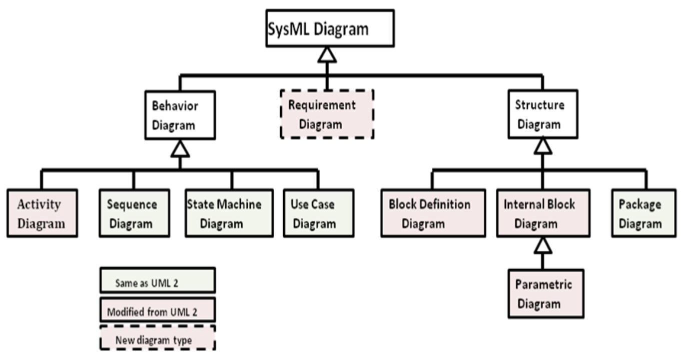

Fig. 5. SysML Artifact Hierarchy and Relationship to UML

Fig. 5 illustrates the types of diagrams used by SysML; there are three primary types of diagrams at the highest level of the hierarchy: behavior, structure, and requirements. The requirements diagram is new and not part of UML. At the next level down, SysML makes specialized modifications to UML activity diagrams and it has made significant modifications to

UML class diagrams, as extended by UML composite structures, to create what are called block diagrams. A completely new type of diagram called the parametric diagram has been introduced, which shows mathematical relationships among the pieces of the system being designed and more specifically combines mathematical formulas for analysis of critical system parameters. Parametric diagrams have a key role in system modeling.

II. HISTORY OF MODELING FOR SOFTWARE AND SYSTEMS

This section provides a brief history of some of the key historical contributions to modeling for software and systems over the past half century. The mathematical concept that a system can be regarded as a nonempty set upon which relations are defined can be traced back as early as Tarski [6]. More generally 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. Lin compiled an extensive survey of the mathematical concepts and literature in [7]. Lin and Ma [8] defined a general system to be an ordered pair (M, R) where M is a set and R is a collection of relations on M, i.e. a relational structure. This definition is a formalization of the concept that a system is a "whole" consisting of interacting "parts", which was expressed by von Bertalanffy [9] as early as 1967 and later by INCOSE [10].

A. Tarski Model Theory

First-order model theory is concerned with the relationships between system descriptions made in a first-order formal language (such as the Predicate Calculus of mathematical logic) and the structures that satisfy these descriptions. Central to first-order models is Tarski's model theoretic definition of truth [6]. Specifically, given a relational structure (M, R), a sentence in the formal language is defined to be true if there is an interpretation of the sentence into the structure for which the sentence becomes true in the structure. The interpretation in this approach is an isomorphism, i.e. the symbols of the sentence are in one-to-one correspondence with the symbols in the image under interpretation into the structure. Instantiation in the sense of Object Orientation can be regarded as this type of interpretation. In Tarski Model Theory the relational structure is said to be a *model of the sentence*.

The geometric examples in Fig. 1 can be written as first order models of the axiom system (sentences) for the elementary geometry of Section I. For example, the planar geometry model can be defined by taking M to be the Cartesian plane and R to be the collection of all (straight) lines in the plane. Specifically, a line ℓ is defined to be a unary relation R_ℓ on a subset of points (x_1, x_2) in the plane which are related by the equation $x_2 = m_\ell x_1 + b_\ell$ where m_ℓ and b_ℓ are the geometric characteristics (i.e. attributes) associated with the line ℓ. The three axioms can be interpreted rigorously into the Cartesian plane using the UML graphical models of the three sentences given in Fig. 4. The truth of the interpretation of each statement is then easily inferred from simple algebraic calculations.

B. Yourdon Structured Analysis and Design

Yourdon [11] introduced a model based approach for software development from a behavioral viewpoint. He also introduced a graphical modeling language called Data Flow Diagrams to support his approach. 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 forms of cohesion are based on functionality and communication. Cohesion in modules minimizes interactions between elements not in the same module, thus minimizing the number of connections and amount of coupling between modules. Yourdon also considered that the solution should reflect the inherent structure of the problem.

Yourdon also introduced the concept of Structured Analysis, which is based on a principle that the specification of the problem should be separate that of the solution. He proposed that separation be accomplished by using two types of models: the Essential Model, which is an implementation-free representation of the system, and the Implementation Model, which is a behavior specific representation. These two types of models are consistent with the concepts of the axiom system which identified the need for two types of system models, one for description or specification and the other for interpretation or implementation. Structured Analysis requires that the concerns of the two types of models be separated.

C. Wymore MBSE

A. Wayne Wymore [12] was one of the early engineering pioneers in the domain of model based systems engineering. He had a behavioral viewpoint on system in which the model of behavior is comprised of the name of the system, its states, the inputs and outputs of those states, and next state transitions. The model also contained a readout function which provided the output of the state 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 will specify 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 models exhibit the same behavior under the mapping. Homomorphism between functional and implementation system design models assures intended system behavior. This is similar to Yourdon's concept of Structured Design that the solution should reflect the inherent structure of the problem.

The Wymore modeling approach to system design was based on the 'tricotyledon'. In a 'garden' for system development, cotyledons are the 'seed leaves' from which systems will eventually 'flower'. Wymore envisioned three types of cotyledons for system design: Functionality; Buildability; Implementability.

Systems analyses are performed using the cotyledons. These include trade-offs between figures of merit such as performance and cost, testing and acceptance of the design and implementation, and how well the design satisfies requirements throughout the life cycle.

D. Klir and Lin General Systems Methodologies

Wymore's concept of homomorphism was generalized by Lin [7] in his concept of a general system by using an algebraic definition of homomorphism. Specifically, given two general system models, $S_1 = (M_1, R_1)$ and $S_2 = (M_2, R_2)$ a mapping h: $S_1 \rightarrow S_2$ is a (relational) homomorphism if for each relation $r \varepsilon R_1$, $h(r) \varepsilon R_2$. It should be noted that in abstract algebra that mapping h is a function and is not permitted to make multi-valued assignments into S_2.

Klir [13] 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 his methodology could be used both for system inquiry (i.e. the modeling of an aspect of reality) and for system design (i.e. the modeling of purposeful man-made objects). Klir's system inquiry can also be regarded as system description.

E. N.P. Suh Axiomatic Design

During the 1990's, N.P. Suh published an extensive body of literature on what he called Axiomatic Design. [14] offers a concise introduction to his approach to include examples. There are two Design Axioms: (i) maximize functional independence by decoupling functional elements and (ii) minimize the information content of the design. The first axiom reflects the principle of structured design prescribed by Yourdon. Axiomatic design uses a flow diagram to concisely represent the system design. A hierarchy of three domains of parameters was core to Suh's approach: functional requirements (FR), design parameters (DP), and process variables (PV). The three domains were linked by two matrices A and B, referred to as the design matrix and the process matrix, respectively. In a linear model using matrix multiplication this gives:

$$[FR] = A\,[DP] \quad \text{and} \quad [DP] = B\,[PV]$$

The functional requirements in this case are expressed as linear combinations of the design parameters with coefficients that could be functions. The Design Equation is then given by $r_i = \Sigma_j A_{ij} p_j$, where the r_i are the specified requirements variables, the p_j are the design variables, and $[A_{ij}]$ is the Design Matrix. The solutions for A_{ij} and p_j are derived from analysis and applying the Design Axioms. Similarly the design parameters are expressed in terms of the process variables.

Modules in the structured design are regarded as rows of the Design Matrix. Independence of the system functions (the first axiom) requires that the design matrix A must be triangular. If A is also diagonal, then each FR can be satisfied independently by one DP, in which case the design is called uncoupled. Otherwise DPs must be changed in proper sequence, in which case the design is called decoupled. Coupled designs consist of all other cases (i.e. not uncoupled or decoupled). Coupled designs violate the Axiom of Independence.

F. Historical Summary

During the past half century key historical contributions have been made to establish model based approaches for systems description, analysis and design. Generally a system has been regarded as a collection of objects (entities) with attributes and relations between the objects as well as relations between the attributes. The mathematical formalization of system modeling has consistently been sought and in principle can be accomplished using the first order model theory of Tarski and the homomorphism of relational structures prescribed by Klir and Yin. But in practice a less formal and more intuitive approach has been followed in software and systems engineering using graphical modeling languages.

The organization of system entities into components which are cohesive modules is a pervasive theme that can be traced from Yourdon to Suh and will be seen to be carried forward into the model based methodologies of the past decade. This is the principle of Structured Design. Equally traceable is the principle of Structured Analysis which states that the specification of the problem should be separate that of the solution. Each of the model based approaches has also incorporated some form of model transformation that preserves structure and behavior. Yourdon sought to preserve the structure of the problem being solved in the structure of the solution. Wymore and Klir sought homomorphic preservation of relations in the flow of events, functions, and states. And the Axiomatic Design of Suh used the Design Matrix to mathematically transform between functional requirements and design parameters and process variables.

III. MODEL BASED METHODOLOGIES

Starting with the INCOSE survey of MBSE methodologies [15], this section provides a summary of the MBSE methodologies and supporting commercial modeling tools that have been developed over the past two decades. These provide a starting point for an overview of the INCOSE MBSE Initiative, which has been ongoing since 2007.

A. *IBM MBSE Methodologies*

Among the IBM methodologies for MBSE are Harmony and the Rational Unified Process (RUP) for Systems Engineering. Harmony is used for integrated systems & software development. Its process elements include: Requirements Analysis, System Functional Analysis, and Architectural Design. The Rational Unified Process for Systems Engineering extends the IBM RUP for software development to systems. Its elements include: Roles, Work Products, and Tasks. Modeling is supported at the level of: Context, Analysis, Design, and Implementation. This methodology is supported by the IBM Rational Suite.

B. *INCOSE OOSEM*

The Object-Oriented Systems Engineering Methodology (OOSEM), developed in the 1990s by the Chesapeake Chapter of INCOSE with significant aerospace involvement, 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 Optimization and Evaluation of Alternatives. OOSEM is supported by any of a number of tools that have been commercially developed for SysML.

C. *Vitech MBSE Methodology*

Based on concurrent systems engineering activities that reflect the Systems Engineering Vee, the activities of the Vitech MBSE Methodology are: Source Requirements Analysis, Functional Behavior 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. It is commercially supported by the Vitech CORE product suite.

D. *JPL State Analysis*

The State Analysis of Methodology was developed by the California Institute of Technology Jet Propulsion Laboratory (JPL) with deep space missions in mind. Using Goal-directed Operations Engineering the methodology 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 modeling. Models are used to describe system evolution. Core tenants include state-based behavioral modeling and state-based software design. The methodology seeks to reduce gaps in software implementation of systems engineering requirements. The JPL State Analysis can augment the Vitech CORE functional analysis schema to synthesize functional and state analysis

E. *Dori Object-Process Methodology (OPM)*

The Dori OPM 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 Object-Process Language (OPL). OPD constructs have semantically equivalent OPL sentences. OPL is oriented towards humans as well as machines. System structural links (relations) and procedural links (behavior). Structural links are similar to UML class relationships (e.g. generalization). Three mechanisms are used for modeling: (i) unfolding/folding which refines/abstracts structural hierarchy, (ii) zooming out/zooming in which exposes/hides details of an object or process, and (iii) expressing/suppressing: exposes/hides details of a state.

F. *INCOSE MBSE Initiative*

As the systems modeling language SysML was maturing to the level of a formally adopted OMG specification, an MBSE Initiative was organized by INCOSE in 2007 for the purpose of establishing MBSE methodologies and integrating them into existing systems engineering practice [16]. A key focus of the initiative is the shift from document centric processes to systems engineering to a model centric processes. Specifically in this initiative, MBSE is envisioned as the formalized application of modeling to support systems engineering beginning in the conceptual design phase and continuing throughout development and later life cycle phases. The MBSE Initiative is currently researching five main themes: (i) modeling and simulation interoperability and how models interact with each other throughout the system lifecycle, (ii) modeling for Space systems, (iii) telescope modeling for the active phasing experiment, (iv) Biomedical device reference architecture, and (v) Global Earth Observation System of Systems (GEOSS) to provide information for decision support tools for a wide variety of users worldwide [17].

G. *Model Driven Architecture (MDA)*

Among the numerous standards and specifications of the Object Management Group (OMG) are UML, SysML, the Meta Object Facility (MOF), and the Model Driven Architecture (MDA^{TM}); which come together in MDA^{TM}. MOF, for example, is a standard for model-driven engineering and is the mandatory modeling foundation for MDA^{TM}.

MDA^{TM} is a significant paradigm shift in software engineering in which the OMG made a dramatic move from their Object Management Architecture to models. Initiated in late 2000 and public since 2001, it is a trademarked term from the OMG. Its standards along with a large body of reference material can be found on OMG open websites such as [18]. MDA^{TM} provides an open, vendor neutral approach to the challenge of business and technology change. It separates business and application logic from underlying platform technology; and seeks to insulate the core of the application. This separation of concerns is an example of Structured Design. The MDA^{TM} specified by OMG accomplishes the separation by specifying UML models of the system and transformations between those models. The OMG MDA^{TM} is currently entering its second generation of specification.

H. *ISO/IEC Standard 42010*

ISO [21] has been conceptualized System architecture through relationships: *System Architecture* is the fundamental conception of a system in its environment embodied in elements, their relationships to each other and to the environment, and principles guiding system design and evolution. The specification of models associated with a system from a relational viewpoint therefore has a natural compliance with the ISO specification of system architecture.

IV. RELATIONAL ORIENTATION

Relational oriented system engineering (ROSE) as introduced in [19] is a general systems methodology that employs a principle of *model specification and relational transformation* for the purpose of system specification, analysis, and design. It is similar to but more formal than the methodology of Klir [13] which rests upon a principle of alternatively using *abstraction* and *interpretation* for problem solving. The ROSE methodology generalizes the functional and hierarchical viewpoint of legacy systems engineering which rests upon a principle of *definition* and *decomposition* for system specification. Furthermore, ROSE extends the concept of relational homomorphism for general system models used by Lin [7] to a multi-valued bidirectional relational transformation that is algebraically computable.

A. *Specification of Models in Relational Orientation*

From the relational viewpoint, the *specification of a model* associated with a system is the specification of:

- Entities associated with the system
- Sentences (declarations) about the entities
- Modeling elements to instantiate the sentences
- A semantic structure on the modeling elements
- Interpretations of the sentences into the semantic structure

The entities of the system can include attributes, classes and components of the system. There can also be entities associated with the system which are not part of it, such as the environment. The sentences are the basis for system specification. They should be complete in that they determine all intended relations or associations between the entities, and also be consistent. The sentences are instantiated by the modeling elements of the specified semantic structure. The model is valid when the interpretation of each sentence is true within the structure. Relational orientation is primarily concerned with two types of semantic structures: relational structures and graphical models. *Relational frames* will be defined in Subsection C and used to specify semantic structures for organizing knowledge about the system.

Modeling elements can have four types of *relational association*: (i) relation by belonging to a defined subset of elements (collection of the model elements), (ii) n-ary mathematical relation, (iii) hierarchical association (decomposition of individual model elements), and (iv) association with elements of another model by transformation. The first three types correspond to the internal structure of the model. The associated relational frames will be referred to simply as *frames*. The fourth type of association is external to the model; this will be referred to as a *transformational frame*.

When the frames of two models of a system are associated by a transformational frame, the collective three frames will be referred to as a *framework*. In relational orientation, systems are modeled using multiple frameworks which represent the various knowledge domains and components of the system. Frameworks are integrated into a *framework structure* by sharing common frames or by transformational associations between frames.

The specification of frames for the models and transformational frames between the models is complete when they form a framework structure that is adequate for system specification, analysis and design. This resultant framework structure provides a metamodel of the system, i.e. an abstract model with rules for specifying the models of the system.

B. *Semantic Structures*

Semantic structure is a concept which seeks to formalize the intended meaning of natural language through some type of organization. This could be as elementary as the 'verb-noun-object' structure of the English language, as prescriptive as the Hyper-Text Markup Language (HTML), or as mathematically precise as a relational structure on a set. Formalization of semantics includes creating a list or dictionary of terms, rules for grammar, and a schema for organization. While semantic structures are a subject of on-going research and definition, they are central to the specification of models from the relational viewpoint. Without semantic structures only the syntax of a model and its association with a system could be specified with precision.

The concern of ROSE primarily with relational structures and graphical models for formalizing semantics is due to the relational viewpoint. Each of these two types of structure organizes the knowledge of the system around specified relations. This is made clear by the Elementary Geometry System (in Section I), the meaning of which is understood through the relations of the key words in the axiom system to each other and through their interpretation into a meaningful model. As depicted in Fig. 4, the axioms admit modeling through the graphical language of UML Class Diagrams that capture the relations in the axioms (sentences). These diagrams are a type of semantic structure that brings precision to modeling the relations in the axioms without changing their meaning. A mathematically defined relational structure (e.g. lines in the Cartesian plane) is another type of semantic structure. It brings meaning through interpretation of the axioms.

C. *Relational Frames*

The concept of a *relational frame* will be defined in support of the semantic structures needed for model specification. Relational frames provide a static structure for organizing knowledge about the system captured by the model using specified relations that reflect the structure of the semantic knowledge captured by the sentences and their interpretation into the relational structure. This is a generalization of the concept of *object oriented frames* used in software engineering as in [20], which are primarily templates that provide structural slots or placeholders for entity descriptions such as the allocation of attributes or methods to a class.

ROSE is concerned primarily with two types of models for system specification, analysis, and design. The first is required for the interpretation of specification (sentences) into a design (model). The other is associated with the intrinsic meaning of the system specification derived exclusively from the relations intended by the sentences comprising the specification. In this case, additional meaning contributed from interpretation is not desirable. Relational frames support the semantic structures needed for each type of these models.

Given a collection *M* of modeling elements and a type of semantic structure R for organizing the relations on the elements, a relational frame <u>M</u> is defined to be the ordered pair (*M*, R). If *M* is a collection of mathematical objects, such as numbers or sets, and R = $\{R_\alpha\}$ is a relational structure on *M*, then the frame (*M*, R) becomes Lin's general system model.

The notation (*N*, S) for frames will be reserved for models of the system that *interpret* the sentences W = $\{W_\beta\}$ used for specification. In this case the key words in the sentences are assigned to designated elements or relations in the modeling frame (*N*, S) and the implications of the assignment must be inferred. The specification of a model associated with the design of a system of will then be denoted as **N** = (*N*, S; W).

When the semantics of the sentences are modeled using a graphical language for the purpose of capturing only the intended relations without introducing (design) interpretation, the modeling elements will become nodes in the graphs and will be denoted as *G*. The semantic structure will be denoted as H = $\{H_\mu\}$. If the elements of *G* are taken to be UML classes, the semantic structure H is given by Class Diagrams, as illustrated in the graphical models in Fig.4. When the sentences W = $\{W_\beta\}$ are written in a language *L* (e.g. English, Predicate Calculus), it is possible to define a relational frame (*L*, W) in the same way. In this case, the modeling elements are the key words from the sentences and the relations between the key words are determined by the direct semantic associations in the sentences. The key words in *L* should be the same as the graphical modeling elements that would be used as nodes of *G*.

All of these types of frames share a common syntax and semantic style that lends itself to matrix representation. Each type specifies modeling elements and a semantic structure for those elements. The modeling elements can be used as the headers of the rows and columns of a square matrix. The semantic structure can be used to fill in the cells of the matrix according to whether two elements are directly associated by the structure or not. In the event that they are the entry to the cell can be a symbol or notation for the semantic of the association of the elements. Matrix representations of frames will be important to formalizing relational transformation.

D. *Syntax and Semantics of Relations and Relationships*

Mathematical relations and relationships as introduced in Section 1 have a precise syntax that is given by unary (i.e. subset) and n-ary associations. The syntax of *n-ary association* is the 'n-tuple'. Strictly speaking these associations can be regarded as mathematical relationships. However, the interpretation of a relationship into a model (relational structure) is intended to give meaning to the syntax. The term *semantic relationship* will be applied an n-ary association that has been given meaning through interpretation into a semantic structure. Mathematical relations are collections of n-ary associations, i.e. the syntax is the unary relation of subset. *Semantic relations* are then subsets of semantic relationships.

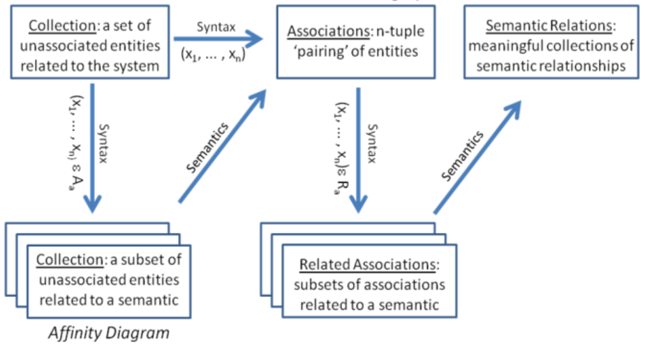
Interrelation Diagraph
Collection: a set of unassociated entities related to the system
Syntax
$(x_1, \dots, x_n)$
Associations: n-tuple 'pairing' of entities
Semantic Relations: meaningful collections of semantic relationships
$(x_1, \dots, x_n) \in A_a$
Syntax
Semantics
$(x_1, \dots, x_n) \in R_a$
Syntax
Semantics
Collection: a subset of unassociated entities related to a semantic
Related Associations: subsets of associations related to a semantic
Affinity Diagram

Figure 6: Syntax and Semantics of Relations in Practice

In engineering practice, however, semantic relations are frequently built up from a list of entities in a different but still systematic way [22, chapter 24], as illustrated in Fig. 6. The initial list is an unassociated collection of entities whose only specified association is that they are in some way related to the system of interest. This list can be generated by formal review processes or by informal 'brainstorming'.

The entities can be grouped according to 'affinity', i.e. subsets of the entities that have a semantic in common (besides the system). The syntax is that of unary association (subsets). For example, if the system were an aircraft; performance, operational effectiveness, and economics can be used as semantics to group some of the entities associated with the system. The performance grouping would include the attributes of range, speed, fuel capacity, and payload weight.

The initial list of entities can also be grouped into n-ary associations by paring the entities using an Interrelationship Diagraph as indicated at the middle top of Fig. 6. When these n-ary associations are integrated into the semantic groupings of the Affinity Diagram, the result is a version of the affinity groupings in which the entities related to the semantics are organized into subsets of associated entities. In the performance grouping of the aircraft system, for example; aircraft range, speed, fuel capacity, and payload weight would be associated into a 4-tuple, which as yet has no semantic between the four attributes but does have the semantic of the grouping ('performance'). Simple semantics such as the decrease of range with increased speed and payload or detailed semantics such as the Breguet range equation must ultimately be included. This final step, which is indicated in the upper right corner of Fig. 6 results in the original affinity groupings becoming semantic relations, i.e. meaningful collections (subsets) of semantic relationships.

E. *Syntax of Relational Transformation*

Relational transformations admit a syntax that can be used for calculation of the transformation of relationships. This section summarizes the calculation of binary and unary relational transformations. Further details can be found in [22].

1) Binary Relational Transformation: Given a model **M**, with elements in the set *M*, a mathematical binary relation R_α on *M* is a collection of pairs of elements y_i, $y_j \in M$ that are associated by R_α as an ordered pair, i.e. $(y_i, y_j) \in R_\alpha$. The equivalent notation $y_i R_\alpha y_j$ is also used.

Let **N** be another model, with elements in *N*, and a binary relation S_β on *N.* A binary *transformational association* Q between **M** and **N** is a collection of ordered pairs of elements taken from *M* and *N*, i.e. $(y_i, x_k) \in Q$. The element $y_i \in M$ is said to be associated with the element $x_k \in N$ by Q. The equivalent notation $y_i Q x_k$ is also used. Transformational association can be multi-valued, i.e. each y_i can be associated with multiple x_k, and is also bi-directional, i.e. the association $(y_i, x_k) \in Q$ is also an association of the element $x_k \in N$ with the element $y_i \in M$.

The *calculation* of the transformation of binary relationships is straight forward:

$(y_i, y_j) \in R_\alpha$ with $(y_i, x_k), (y_j, x_l) \in Q$ implies $(x_k, x_l) \in R_\alpha Q$.

If $R_\alpha Q$ is a subset of one of the relations on N, e.g. $R_\alpha Q$ is a subset of S_β, then Q is said to induce a *relational transformation* $\mathbf{M} \rightarrow \mathbf{N}$. In the special case when Q is determined by a function, q: $M \rightarrow N$ (i.e. not just a transformational association), then the equation above means for $(y_i, y_j) \in R_\alpha$ then $(q(y_i), q(y_j)) \in S_\beta$, since $R_\alpha Q$ is a subset of S_β. Thus, q is a relational homomorphism in the sense of the general systems theory of Lin [7]. This demonstrates that relational transformation is a generalization of the relational homomorphism used by Lin.

Relational transformations have a broad range of applications, one of which is to bring precision to the interpretation of the sentences used for system specification. Let **N** = (*N*, S; W) be a model associated with a system of interest that has been specified from the relational viewpoint. The semantic structure S of a proposed design, although not specified independently of the specification W, is not a direct interpretation of W. Rather W must be interpreted into S. (This is just like the interpretation of the elementary geometry axiom system into the semantic structure of the planar geometry.)

Precision can be added to the interpretation of W into S by using graphical models and relational transformations. First let (*G*, H) be the frame that graphically models the relations of the sentences W. The semantic structure H = $\{H_\mu\}$ could, for example, be a collection of UML Class Diagrams (as was the case with the elementary geometry axiom system). Let Q be the binary transformational association that associates each graphical modeling element $g_i \varepsilon G$ (e.g. a UML Class) with an element $x_k \in N$. The binary transformation $\mathbf{G} \rightarrow \mathbf{N}$ results in a relational structure HQ = $\{H_\mu Q\}$ on *N* that can now be compared rigorously to relational structure S for validity.

2) Unary Relational Transformation: Finally, of special interest are unary transformations, which associate subsets between domains. If Q associates *M* with *N* and if R_α is a subset of *M*, define $R_\alpha Q = \{x \in N\text{: } yQx \text{ for some } y \in R_\alpha\}$.This is the subset of *N* that Q has associated with the subset R_α of *M* and will be referred to as the *unary transformation* of R_α by Q. Unary transformation is based on the mathematically natural binary relation defined by subset relationship. Unary transformations are useful for association of data in tables.

F. *Types of Relations Transformed*

There are many ways that the elements or parameters of a model can relate to or depend upon each other. The types of relationship can range from logical to metric. These include precedence order, client-server dependencies, and sensitivities derived from simulation and analytics. Statistical correlation is another type of relationship that admits transformation. There are both analytical and numerical methods for computing the transformation of correlation and other statistical quantities. Relational frames are well suited to capture dependencies in a style similar to but more general than design structure matrices.

V. Relational viewpoint on design and verification

In the practice of engineering development, Design and Verification can be substantially separated. In the legacy Systems Engineering V, for example, System Level Design can be separated from System Level Verification by two levels of specification plus testing. In a model driven approach, the focus is shifted from verification of designs by physical testing to verification of the design models.

The model driven approach of Relational Orientation will inherently add a benefit of greater concurrency of verification with design. The same models used for design can also be re-used for verification. This will be demonstrated by applying Relational Orientation to the N.P Suh approach for Axiomatic Design. The approach from the relational viewpoint begins with an engineer seeking to improve an existing design who uses the Geometry System to perform design and analysis (as illustrated in the Use Case in Fig.3) to accomplish this task. The engineer will use a *model* of the Geometry System to apply a local optimization algorithm to explore the Design Space. This is representative of how the engineer in a model driven environment would behave in general, i.e. to improve the design of any system component; the engineer would access a model of the component. Thus, Relational Orientation supports access to the models of a system in much the same way as UML Classes access each other's software objects.

A. *Applying Relational Orientation to Axiomatic Design*

A relational orientated local optimization algorithm for design improvement will be specified for linear Axiomatic Design problem and demonstrated in three dimensions. The conventional solution is to solve the design equation subject to an objective value for each requirement. Unlike the conventional solution the algorithm will seek acceptable ranges of values to meet the requirements. Furthermore, in Axiomatic Design the variables in each vector are intended to be independent of each other, but in practice constraints will create dependencies between design variables. The algorithm must also account for these dependencies.

In relational orientation, the components of the vector [FR] become the entities of a relational frame $\underline{F}$ for the Requirements Space. The components of [DP] become the entities of a relational frame $\underline{D}$ for Design Space. For Axiomatic Design, the conventional problem is to solve the design equation: [FR] = A [DP], where A is the design matrix. In Relational Orientation, the transformational frame will be a matrix Q_A of the sensitivities of the requirements parameters to the design variables. The three frames $\underline{F}$, $\underline{D}$, and Q_A comprise a relational framework for the design problem. Fig. 7 illustrates a full framework structure for the design and process problems of Axiomatic Design.

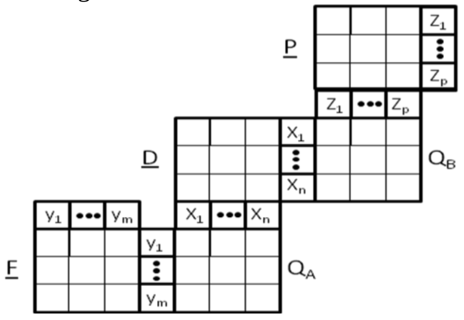

Figure 7: Framework Structure for Axiomatic Design

B. *Achieving Concurrency in Design and Verification*

The framework structure for linear Axiomatic Design (FSLAD) and the design equation can be used for design and for verification, respectively. The design framework and the design equation are just different views of the same information. The design matrix A in general contains the equations $y_i = f_i (x_1, \ldots , x_n)$, which are the response surfaces used to test the system response for specific values of the design parameters against requirements specifications such as $y_i \geq Y_i$. In conventional design approaches, requirements are allocated to a system function or component. The design engineer then proposes a solution $(x_1^*, \ldots , x_n^*)$ and tests the response y_i^* against the requirement Y_i using the response surface $y_i = f_i (x_1, \ldots , x_n)$. The performance margin is given by $Y_i - y_i^*$, which is a measure of robustness.

In relational orientation, a framework such as $\underline{F}$-Q_A-$\underline{D}$ in FSLAD can be used to search the Design Space for improvements in functionality or robustness. The metric sensitivities Q_A and $\underline{D}$ are used to navigate the solution space. Specifically, the differentials of the response surface guide the way to improvement in the Design Space:

$$\frac{dy_i}{dx_l} = \sum_{k=1}^{n} \frac{\partial f_i}{\partial x_k} \frac{\partial x_k}{\partial x_l}$$

If the design parameters x_k are independent then there is no summation in the above expression. However, if there are any joint constraints on the design parameters then the partial derivatives between the parameters will contribute more than one term to the summation. The search algorithm in the next subsection is developed around the above expression. It is similar to a Steepest Descent Method but not the same due to the effects of joint constraints. It also shares similarities with but is distinct from the linear programming Simplex Method. The style and notation should support further research into more advanced methods. In this approach no candidate designs are considered that are not already solutions. The design matrix response surfaces are only used to test robustness.

C. *Searching the Solution Space in FSLAD*

We shall assume that the Design Space is a convex region constrained by (linear) equations on the design parameters. The system designer explores the space for robust solutions that satisfy specified requirements. Because the space is convex, a preferred set of paths can be used for the search. These are the edges of the convex hull of the space. In an N-dimensional solution space, an edge is a line formed by the intersection of N-1 linearly independent constraint surfaces. Each edge has only two possible directions of change. Each vertex is the intersection of N linearly independent surfaces.

The procedure for local search for design improvement against one specified requirement *y* is as follows:

1. Begin with an initial solution $\underline{s}_0$ that lies on one of the edges. If the solution is on a vertex then go to step 4.
2. Navigate the edge in the direction of improvement to a vertex by: (i) computing the directional derivative $dy/d\underline{u}$ along the edge, where $\underline{u}$ is the unit vector defining the edge, and (ii) choosing the direction $\underline{u}_0$ of improvement ($+\underline{u}$ or $-\underline{u}$) indicated by the directional derivative.
3. Solve for the vertex by: (i) extending the unit vector $\underline{u}_0$ to a ray $\underline{r} = \underline{s}_0 + \lambda\underline{u}_o$ to intersect any possible constraint surface not comprising the edge and (ii) choosing the closest surface that intersects the ray. This intersection point (formed from N equations) is the vertex which is the next candidate for solution improvement.
4. Find all edges that intersect the vertex by: (i) evaluating the equation of each remaining constraint surface (i.e. *not* one of the N equations that produced the vertex) at the vertex and (ii) for each intersecting surface, form new edges by deleting one of the previous surfaces and replacing it with the new surface. In the case of multiple surfaces, multiple replacements are admissible.
5. Find and navigate the direction of greatest improvement by: (i) computing directional derivatives along each new edge and (ii) repeating step 3 to generate the next vertex.
6. The procedure is over when the requirement is met robustly or no further improvement can be achieved.

The constraint surfaces in Fig.8 will be used to illustrate the algorithm and procedure.

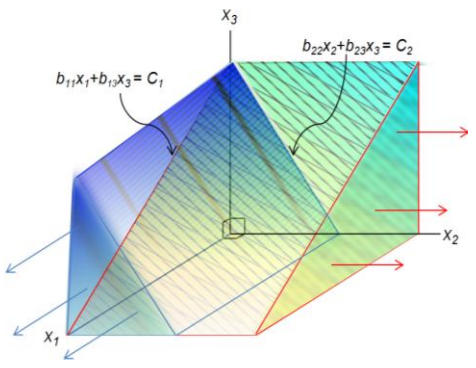

Fig.8. Constraint Surfaces for a 3-dimensional Design Space

In the relational viewpoint, the Requirements Model is one-dimensional with one relation: $dy > 0$. The Design Model has a structure of five relations given by the constraints in Fig. 8 and be represented by:

$$x_1 \geq 0 \qquad (S_1)$$

$$x_2 \geq 0 \qquad (S_2)$$

$$x_3 \geq 0 \qquad (S_3)$$

$$b_{11}x_1 + b_{13}x_3 \leq C_1 \qquad (S_4)$$

$$b_{22}x_2 + b_{23}x_3 \leq C_2 \qquad (S_5)$$

D. *Using Sensitivities to Find the Solution*

The feasible region of the Design Model is convex because the coefficients b_{11}, b_{13}, b_{22}, *and* b_{23} are positive. It can be derived from the two surfaces in Fig.8 and represented by a four sided pyramid as illustrated on the left in Fig.9. Navigation occurs along one of the eight edges of the pyramid. On the right of Fig. 9 is a 2-dimensional view of the surfaces of the pyramid in the x_2-x_3 plane within the feasibility region.

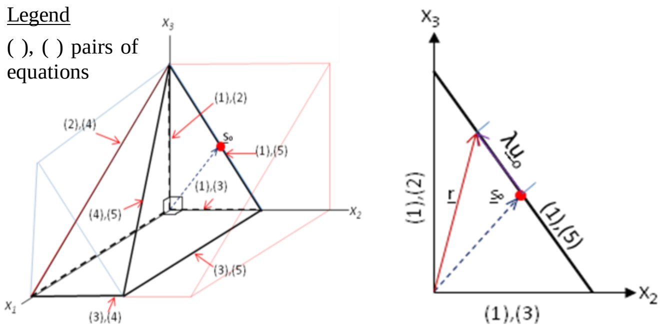

Fig.9. 3-dimensional Feasibility Region

The feasibility region is determined by E constraints. Any given solution $\underline{s}_o$ that lies on an edge within the region must satisfy: (i) all (E) constraint inequalities and (ii) 'N-1' constraint equalities (N is the number of dimensions). In this case N=3, therefore edges are formed from pairs of the (E = 5) equalities.

From the position $\underline{s}_o$, navigate the edge in the direction of improvement by computing the directional derivative $dy/d\underline{u}$, where $\underline{u}$ is the unit vector for the direction of the edge; thus $\underline{u}_o$ is to be $\pm\,\underline{u}$ depending on which one corresponds to $dy > 0$.

Extend the vector $\underline{r} = \underline{s}_o + \lambda\underline{u}_0$ and calculate $\underline{r}$ by extending the ray to the intersection with the other constraint surfaces and then substituting $\underline{r}$ into the equation of the surface and solving for λ. By convexity, $\underline{r}$ will belong to the feasible region only for the closest surface. This corresponds to the minimum value of λ over all possible intersections. This value will be denoted as λ_{min}. The calculation of λ_{min} determines the vertex:

$$\underline{s}_i = \underline{r} = \underline{s}_o + \lambda_{min}\,\underline{u}_0$$

There may be multiple surfaces $S_{n1},\ldots, S_{n*}$ that each have $\lambda = \lambda_{min}$. To find all edges that intersect the vertex; each new edge can be formed by replacing a previous edge surface with a new surface (e.g. S_1, S_2 is a new edge formed from S_1, S_5). This replaces one design variable with a new design variable. If $\lambda = 0$ or ∞ (i.e. $\underline{r}$ is unconstrained on the edge) the edge is not navigable ($\lambda = 0$) or feasible ($\lambda = \infty$). Otherwise λ obtains a value that is within the feasibility region, and the algorithm can navigate the feasible edges to the next vertex (or vertices).

If $dy < 0$ in all possible new edge directions, at any time during the search process, then the search must be stopped, as the vertex is globally optimal. All of the edges need not be navigated if each of the surfaces (E number of equations) is discarded after its first use. For design verification of a solution $\underline{s}^*$ (yielding the response $y^* \geq Y$), only the design margin $Y - y^*$ must be checked for robustness, which is done with a local edge analysis.

VIII. SUMMARY AND CONCLUSION

Rooted in the first-order model theory of mathematics and practiced extensively in science, the concepts and methods for model based approaches to software and systems engineering have evolved over the past five decades to a level of maturity that is now commercially supported. The fundamental principles of Structured Design and Analysis developed in the early years of software engineering still apply today and are evident in MDA. The emergence of SysML and MBSE methodologies offer the promise executable behavioral models which are critical to system design and analysis. Machine readable SysML and UML models are envisioned to ultimately support services for systems engineering processes.

The ROSE formalism and methodology integrates and extends the legacy work of Yi Lin and George Klir on the use of relational structures and homomorphism to model systems. Hierarchical paradigms such as '*definition and decomposition*' can be expressed more precisely by the ROSE principle of '*model specification and relational transformation.*' The mathematical foundation for ROSE supports the rigorous development of structures for the design of systems and the assemblage of systems of systems and extends the methods of N.P. Suh on axiomatic design theory. Relational Orientation offers a coherent mathematical foundation for ROSE.

The relational approach in this paper is currently being used in a research council sponsored project on open architecture for aviation SoS and is also foundational to a new five year program in the UK for advanced manufacturing.

REFERENCES

[1] S. Hawking, *A Brief History of Time*. Bantam Books, 1988.

[2] P. Chen, "the entity-relationship model – toward a unified view of data", *ACM Transactions on Database Systems*1: 1: 9-36, 1976.

[3] J. Holt, *UML for System Engineering*, 2nd edition, The Institute of Electrical Engineers, London, 2007.

[4] C. Raistrick et al, *Model Driven Architecture with Executable UML*, Cambridge University Press, Cambridge, 2004.

[5] S. Friedenthal, *A Practical Guide to SysML*, Morgan Kaufmann OMG Press, Amsterdam, 2008.

[6] A. Tarski, "Contributions to the theory of models I, II, II," Nederl. Aka. Wetensch. Proc. Ser. A., Vol 57, pp. 572-581, 582-588, Vol 58, pp. 56- 64, 1954, 1955.

[7] Y. Lin, *General Systems Theory: A Mathematical Approach,* Kluwer Academic/Plenum Publishers, New York, 1999.

[8] Y. Lin and Y.-H. Ma, "Remarks on analogy between systems," Int. J. General Systems, Vol 13, pp. 135-141, 1987.

[9] L. von Bertalanffy, "General systems theory: Application to psychology, social science," Infor. Sci. Soc., Vol 6, pp. 125-136, 1967.

[10] International Council on Systems Engineering, *INCOSE Systems Engineering Handbook, v.3.2,* 2010.

[11] E. Yourdon, *Modern Structured Analysis*, Yourdon Press, Upper Saddle River, New Jersey, 1989.

[12] A. Wymore, *Model-Based Systems Engineering*, CRC Press, 1993.

[13] G. Klir, *Facets of Systems Science*, Plenum Press, New York, 1991.

[14] N.P. Suh, "Axiomatic Design Theory for Systems", *Research in Engineering Design*, Vol 10, pp. 189-209, Springer-Verlag, 1998.

[15] J. Estefan, INCOSE Survey of MBSE Methodologies, INCOSE TD 2007-003-02, Seattle, 23 May 2008.

[16] S. Friedenthal, R. Griego, and M. Sampson, "INCOSE Model Based Systems Engineering (MBSE) Initiative", presentation to the INCOSE International Symposium 2007, San Diego.

[17] http://www.omgwiki.org/MBSE/doku.php

[18] http://www.omg.org/mda/

[19] C. Dickerson and D. Mavris, "Relational Oriented Systems Engineering (ROSE): Preliminary report," *Proc. IEEE SoSE '11*, Albuquerque, 2011.

[20] D. Kirk, M. Roper and M. Wood, "Identifying and Addressing problems in object-oriented framework reuse", Empirical Software Engineering, No3 Vol.12, June 2007.

[21] International Organization of Standards (ISO), "Architecture Description*", ISO/IEC Standard 42010 JTC1/SC7/WG42*, 2010.

[22] C.E. Dickerson and D. N. Mavris, "Architecture and Principles of Systems Engineering", *CRC Press Auerbach Publications*, Boca Raton Florida, 2009.

[23] M. Danilovic and B. Sandkull, "The use of dependence structure matrix and domain mapping matrix in managing uncertainty in multiple project situations," *International Journal of Project Management*, Vol. 23, No. 3, pp. 193-203, 2005.

Charles Dickerson (M'06) received the Ph.D. from Purdue University, West Lafayette, IN, in 1980.

He is the Royal Academy of Engineering Chair of Systems Engineering at Loughborough University in the U.K. His research is focused on model driven architecture and systems engineering. He has authored numerous papers as well as a graduate level textbook with the Georgia Institute of Technology on systems engineering and architecture. He has also co-authored an internationally recognized book on military systems architecture as well as key government reports. As a member of the IEEE, the Chair of the INCOSE Architecture Working Group, and Chair of the Mathematical Formalisms Group at the OMG, he works with the systems engineering community on systems architecture practice and standards.

Before joining Loughborough University, he was a Technical Fellow at BAE Systems, providing corporate leadership for architecture-based and system of systems engineering. He has also served as Aegis Systems Engineer for the U.S. Navy Ballistic Missile Defense Program and later as the Director of Architecture for the Chief Engineer of the U.S. Navy. Previously, as a member of MIT Lincoln Laboratory, he conducted research and tests on electromagnetic scattering. His aerospace experience includes air vehicle survivability and design at the Lockheed Skunk Works and at Northrop Advanced Systems, and operations analysis at the Center for Naval Analyses.