What Is the Smallest Model of a System?

William D. Schindel
ICTT System Sciences
schindel@ictt.com

Abstract. How we represent systems is fundamental to the history of mathematics, science, and engineering. Model-based engineering methods shift the nature of representation of systems from historical prose forms to explicit data structures more directly comparable to those of science and mathematics. However, using models does not guarantee simpler representation--indeed a typical fear voiced about models is that they may be too complex.

Minimality of system representations is of both theoretical and practical interest. The mathematical and scientific interest is that the size of a system's "minimal representation" is one definition of its complexity. The practical engineering interest is that the size and redundancy of engineering specifications challenge the effectiveness of systems engineering processes. INCOSE thought leaders have asked how systems work can be made 10:1 simpler to attract a 10:1 larger global community of practitioners. And so, we ask: What is the smallest model of a system?

Introduction and Background: Size Matters!

Representation Size, Purpose, Traditions. This paper discusses possible (and potentially least) upper bounds on the sizes of effective representations of systems, *for the purposes of systems engineering*. Compared to traditional systems engineering approaches, it draws more directly on scientific traditions for representing behavior as physical interaction. Systems engineering is still young, and its connections to supporting sciences is still evolving rapidly.

Language and Compression. This subject may appear to be related to the language used to describe systems, and an interesting thread in the mathematical study of description length is whether minimality is in a sense independent of language (Chaitin, Grunwald, Li and Vitany). In any case, systems modeling languages such as SysML® and its predecessors provide valuable assets for the movement to model-based methods (SysML Partners). Our subject here is not the machinery of these specific modeling languages, but the systems ideas that minimal models must address. When used for system families (product lines, ensembles), the representation described here is subject to significant compression by the use of patterns. This turns out to provide powerful insights about approaches to major practical reductions in the size of SE descriptions and processes, and about ongoing future evolution of domain languages over time. These dynamics also suggest that such patterns can be understood as emergent when the interaction rules of the systems engineering process are properly arranged.

Practical representation challenges of traditional systems engineering. Traditional system documentation of concept of operations (CONOPS), system requirements, design specifications, failure mode and effects analysis (FMEA), test plans, operations and maintenance procedures, and other task-specific system representations over the life cycle of a system can exceed thousands of pages. This does not encourage the engagement of a 10:1

larger global community of systems practitioners. Systems engineers may argue that system risks justify these extensive descriptions, but the effectiveness of these representations may be questioned in light of the following typical experiences: A requirements document, read by three systems engineers, produces three interpretations of its meaning--quite a different experience from three electrical engineers interpreting a properly constructed electrical schematic diagram. Whereas the discovery of an ambiguity in a schematic "blueprint" is considered exceptional (or even machine-checkable in some cases) ambiguities in "system" requirements documents are commonplace and frequently tolerated as the state of the systems art. Determining completeness and consistency of (or otherwise interpreting) a specification document is frequently a highly subjective assignment, requiring very experienced human reviewers. Model-based representations are a hoped-for way to address this challenge, but it is not yet obvious whether these are sufficient for an order-of-magnitude positive benefit to the overall systems process.

SE process versus SE data. This paper's perspective will shift between the systems engineering process (a system in itself), versus information about the target system (which flows through the SE process system), and how the two are related. The systems engineering process is frequently described [ANSI/EIA-632-1998, ISO/IEC 15288 2002, Haskins 2010], but the system representations it produces and consumes (our main subject here) remain a key challenge. We argue that the target systems information is the more fundamental issue to solve, after which the resulting implications for the SE process can be addressed in a new light.

Complexity science. Complexity, a seemingly intuitive idea, has become the subject of formalization and study, including both the natural and human-engineered world. Initial efforts sought a theoretical basis for expressing complexity measures or otherwise understanding complexity, including the size of minimum system descriptions [Li and Vitany 1997, Chaiten 2005, Kauffman 2000]. They have more recently turned to the practical implications of emergent complexity science for engineering processes [Bar-Yam 2003b, 2005, Braha et al 2006, Kuras and White 2005, Schindel 1996]. Some efforts have studied minimal information required to describe a system, as a measure of its complexity. Others have introduced "complex systems engineering" (CSE) terminology in connection with understanding engineering problems or classifying systems, in situations such as highly interconnected systems (networks), adaptive systems, systems embedding humans, issues of scale and scope, ideas about types of emergence, or engineering project failures [Braha et al 2006, Bar-Yam 2003b]. Some studies have focused from the outset on the problems of human engineering or other organic intentional processes in connection with complex engineered systems [Ashby 1957, INCOSE HSIG]. There is a growing awareness of connections between systems engineering and systems science. INCOSE formed the System Science Enabling Group, and later the Systems Science Working Group [INCOSE SSWG], in recognition of the connection between systems science and systems engineering.

System patterns. Ideas of "patterns" have a number of connected roots in science and engineering. Pattern recognition and classification have a mathematical theory and engineering practices [Duda 2001]. Patterns in engineered systems were recognized in building architecture, later inspiring software engineers, and more recently systems engineers. [Alexander 1977, Gamma et al 1995, Haskins 2005, Cloutier and Verma 2007, Schindel 2005b]. Initially expressed using traditional engineering structures (e.g., prose templates), patterns were later combined with model-based systems engineering (MBSE) to lead to pattern-based systems engineering (PBSE) [Schindel and Smith, 2002, Schindel 2005b].

Constructing Effective and Efficient Representations

Using Models. Model-based representations have a traditional engineering role in verifying that designs will satisfy requirements, or otherwise representing system behavior [Karayanakis 1993]. More recently, model-based representations have been used to represent system requirements [Mellor 2002, INCOSE MBSE, Schindel 2005a, SysML Partners, Estafan]. In the earlier and more established design verification case, "model" frequently refers to mathematical descriptions of system physical make-up, often modeling from first principles to create mathematical descriptions that can be analyzed or simulated. In the more recent system requirements case, "model" extends this idea to describe desired functional behavior.

In both cases, the term "model" means a *formal* (according to agreed upon rules), *explicit* (core content not implicitly depending on other assumed knowledge), and *unambiguous* (not subject to multiple interpretations) description. As shown in Figure 1, there are three components in a model-based engineering setting: The model, the system modeled, and the model interpreter(s). We want the model to be interpreted with desired process outcomes (e.g., easy, consistent, and unambiguous interpretation, optimality of design, etc.). Global efforts [ISO 10303 AP233; Mellor 2002] are working toward the exchange and interpretation of model data by machines and people, for purposes of simulation, procurement, fabrication, code generation, etc.

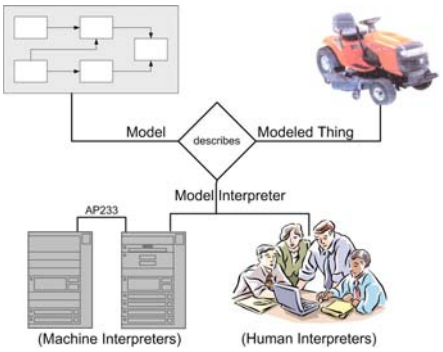

Figure 1: The setting for model-based systems engineering

The "third role" (Model Interpreter) in Figure 1 has vital significance here. The effectiveness of a model means how well it serves the purposes of the Model Interpreter. If we expect to engage a 10:1 larger community of systems practitioners, and make the systems process 10:1 easier, then we must learn how to make the Model Interpreter's tasks easier and more appealing, and for a much larger global population. If we only develop automated approaches to deluge the human Model Interpreter with information, we won't have the outcome needed.

A metamodel. A metamodel is a model of other models—a framework or plan governing the models that it describes. We utilize the S*-metamodel (summarized by Figure 2), a relational / object information model used in the Systematica™ methodology to describe requirements, designs, and other (verification, failure analysis, etc.) information in S*-models. These may be represented in SysML™, database tables, or other languages. We have applied these to systems engineering in mil/aero, transportation, communication, medical and health care, consumer products, construction, manufacturing, and as a framework for educating new engineers [Gunyon et al 2010, Bradley et al 2010, Schindel and Smith 2002, Schindel 2002, 2005b, Ahmed et al 2011].

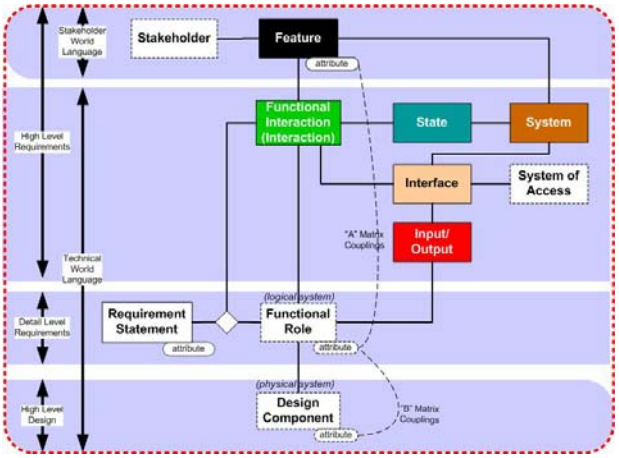

Figure 2: A summary view of the S* metamodel

S*-models describe the external (black box) behavior of target systems twice—once in the subjective (stakeholder) language of stakeholder-valued behavior, and again as more objectively-described technical behaviors.

Stakeholders Features. S*-models represent system stakeholder features as explicit objects. For example, some of the features of an Oil Filter are represented in Figure 3:

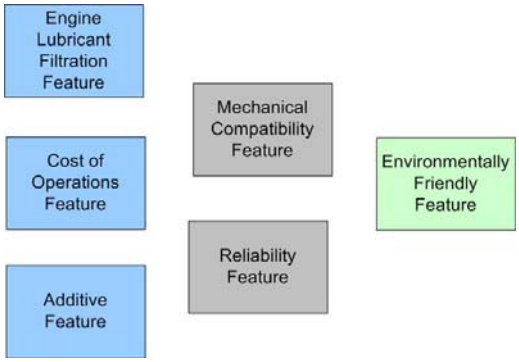

Figure 3: Features of an Oil Filter

We could simply claim that a minimal model of an engineered system must include a (feature) model of all the things valued by all the system's stakeholders. However, there is more to this than meets the eye. Features have a way of creeping into many different engineering conversations and artifacts, not always recognized for their redundancy. Note that every design decision, every trade-off, every value engineering or project argument should ultimately depend upon no less than, *but also no more than*, the Feature model. (This is based upon the practice of including all significant stakeholders and their features in the Feature model.) If we find a compelling argument for why technology X or architecture Y is the right (or wrong) choice, the reason why can only be to better accommodate the Stakeholder Features—trade space is exclusively "scored" in the metrics of these features. A common mistake is to defend choices in technical trade-off spaces that are short of the actual Stakeholder Feature space.

Suppose further that we are performing an FMEA. It turns out that the only "effects" (the E part) that can appear in an FMEA are *failures to deliver on the promise of a stakeholder feature*. As soon as we know the feature space of system, before a design has been synthesized, we can already fill out the "effects" column of the FMEA analysis. (Schindel, 2010) However, it is not universal practice to align or audit FMEA and stakeholder feature models.

Feature space is integrated with technical requirements space by the negotiation of the Features-Interactions relationships—to begin with, a two-column table negotiated jointly by representatives of the stakeholders and the technical community. Feature space is typically of lower dimension than the more technical spaces of system requirements or design. This means that once we have constructed an integrated Feature-Interactions-Roles-Requirements model (traced by Figure 2), we can "configure" (automatically populate) good starting point draft requirements configurations: populating lower dimension features can "automatically" populate higher dimension requirements through the constraints of the model.

We have repeatedly seen the use of Feature models dramatically improve alignment and facilitate constructive discussion of cross-functional teams. For example, a powerful use of Feature space is to express impact assessments on the introduction of new technologies into operations environments, or to express system long range or facility master plans first in terms of features (stakeholder capabilities) planned and only second in terms of the equipment, technologies, or projects that will implement them. Likewise, risk to stakeholders (whether financial risk, schedule risk, technical risk, or otherwise) is represented by Features.

All this suggests that Feature models are often under-utilized in the rush to technical requirements. Note that Feature models are formal even though they are in the (subjective) language of stakeholders. There is a difference between informally stated stakeholder "needs" (in the original voice of the customer) and formally translated (but still stakeholder language and concept) Features. A quick pass through "stakeholder needs" on the way to technical requirements is less than the minimal Features model we are suggesting here.

Feature space is an interesting place. It is the gateway to other communities beyond our engineering organizations, and for that reason may be seen as a strange or unfamiliar language and environment. But, it represents improved connection to those who pay the bills, buy the products, or whose lives depend on the engineered system. Bridging this cultural gap may be challenging, but is the reason that S* Models are dual-rooted in both of the "two cultures". (C.P. Snow, S. J. Gould). When INCOSE thought leaders advocate that we look for ways to engage order-of-magnitude larger segments of the global community in systems work, modeled Feature Space, in model views appropriate to the viewers, are a related enabler.

Interactions. S*-models represent physical interactions as explicit objects at the very core of systems engineering. For example, Figure 4 shows Interaction objects for an oil filter—these summarize the physical interactions of an oil filter with its environment, over its life cycle.

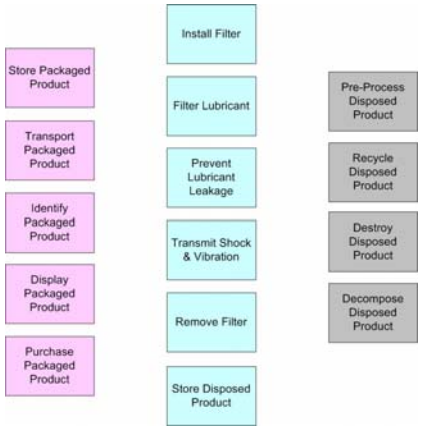

Figure 4: Interactions of an Oil Filter

Interaction models exist at two levels of detail. The High Level Interaction Model simply consists of the name and definition of the interaction, a list of the parties that participate (play roles) in the interaction, and the major attributes of the interaction. These named Interactions also appear within the system's State Model, and that combination very compactly expresses the overall moding of the system's behavior with its environment, over its life cycle. The Detail Level Interaction Model includes an Interaction Diagram (of which there are many specific forms in SysML or other modeling languages) for each interaction, showing the input-outputs exchanged between the interacting actors, and including the requirements statements that describe the roles in the form of "non linear transfer function" relationships between the inputs and outputs (Schindel, 2005). Refer to Figure 5:

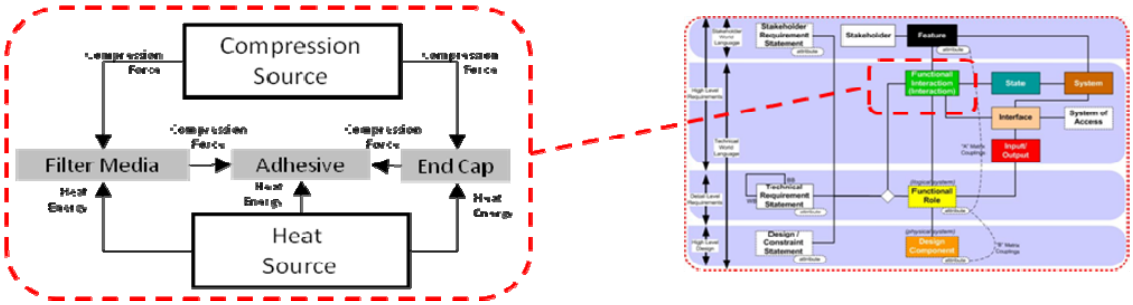

Figure 5: Interaction Diagram

Interaction models go to the heart of what we mean by "system" in the engineering and scientific world, and expresses ideas of emergence. By "system", we mean a collection of interacting components. By "interact", we mean that one component impacts the state of another component. By "state", we mean a property of a component that impacts its current or future behavior. By "behavior", we mean a component's interactions with other components. This is the intentionally circular, relational perspective of the trained scientist, engineer, or mathematician that has helped describe the natural world since Newton. In this perspective, an interaction is holistic, with two or more components playing logical roles. The "emergent" properties of the interaction are associated with the whole, not any single component. Behaviors of individual components are described by requirements statements as input-output characteristics of their roles [Schindel 2005a]. Notice the difference in perspective of Figure 6:

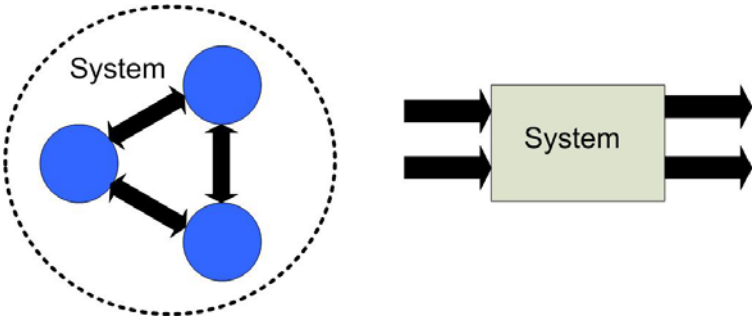

Figure 6: Two Different Starting Points:
Systems As Interacting Components versus A SIPOC Perspective

By now it is well-known that simple behaviors by individual components (or "agents", in the popular parlance), when they interact with each other, may lead to "emergence" of surprisingly more complex behavior by the combined system (e.g., the three body problem, cellular automata, swarms, traffic, etc.). The difference between the simple "rules" (behavior of the actors) and the more complex emergent system behavior is nothing more and nothing less than the difference between describing a functional role in an interaction and the interaction as a whole—it is a difference of night and day.

We have repeatedly observed a profound practical difference between systems engineering modeling interactions as illustrated by Figures 5 and 6 (left) above, versus modeling of SIPOC required behavior as in Figure 6 (right) above. We have seen this difference have major practical impact in numerous SE projects, in which the modeler either did or did not model the whole interaction, including "what the operator did" (Schindel 2006), "what the material did" (Schindel 2011), or other actor behaviors.

Minimality of representation. The S* metamodel arose over time from the research question, "What is the smallest amount of information required to describe system level requirements and design?", combined with practice application. We won't reproduce here the formal argument for minimality of S* models--interested readers may contact the author. However, a summary of that argument is:

- The sufficiency of S* models of requirements and designs is argued, with respect to intended use of the information. Here the *uses of systems engineering information* enter, including considerations of risk and opportunity.
- The minimality of S* models is established by showing that no metaclass (see Figure 2) of information in an S* model is redundant with information in another metaclass, and showing that omission of any component results in loss of sufficiency—including classes versus instances.

This argument makes use of a mapping of which S* model components (grouped across the top of Table 1) are needed for the different SE process areas (summarize by the Table 1 rows):

Table 1: SE Process Areas vs. Metamodel Information Areas

SE Area	Grp1	Grp 2	Grp 3	Grp 4	Grp 5
HLR	X				
DLR/BB	X	X			
DLR/WB	X	X			
HLD	X	X	X		
FMEA	X	X	X	X	
TST	X	X	X	X	X

This table can be constructed for the various SE process areas of ISO15288 or the INCOSE SE Handbook. However, later in this paper we will also discuss an alternate way to view SE process areas. (Why would we want to do that? The answer depends on whether we expect a much larger global population to become traditional systems engineers and take up the traditional SE processes.)

The above minimality argument is “constructive”: Rather than arguing that a minimal model exists, we actually construct it--not the case in most algorithmic information theory. However, this argument does not assert uniqueness: There may be other models no larger that also represent the same system.

Model view; useful redundancy.

A familiar challenge is that different “SE Documents” may be inconsistent with (contradict) each other: This is because they contain redundant information. As documents evolve, that consistency must be maintained to be consistent across the documents. Refer to Figure 7. This issue also occurs within single documents (self-consistency). There are good (task-oriented) reasons why these documents should be redundant—but not why they should be inconsistent.

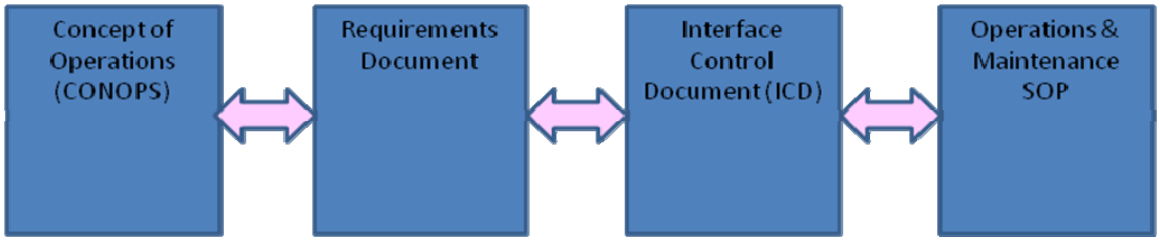

Figure 7: Redundant Documents—Consistent or Inconsistent?

This is one reason why database tools are powerful in systems engineering. Properly used, they can generate different “views” (documents, etc.) from the common underlying data model, thereby improving their consistency:

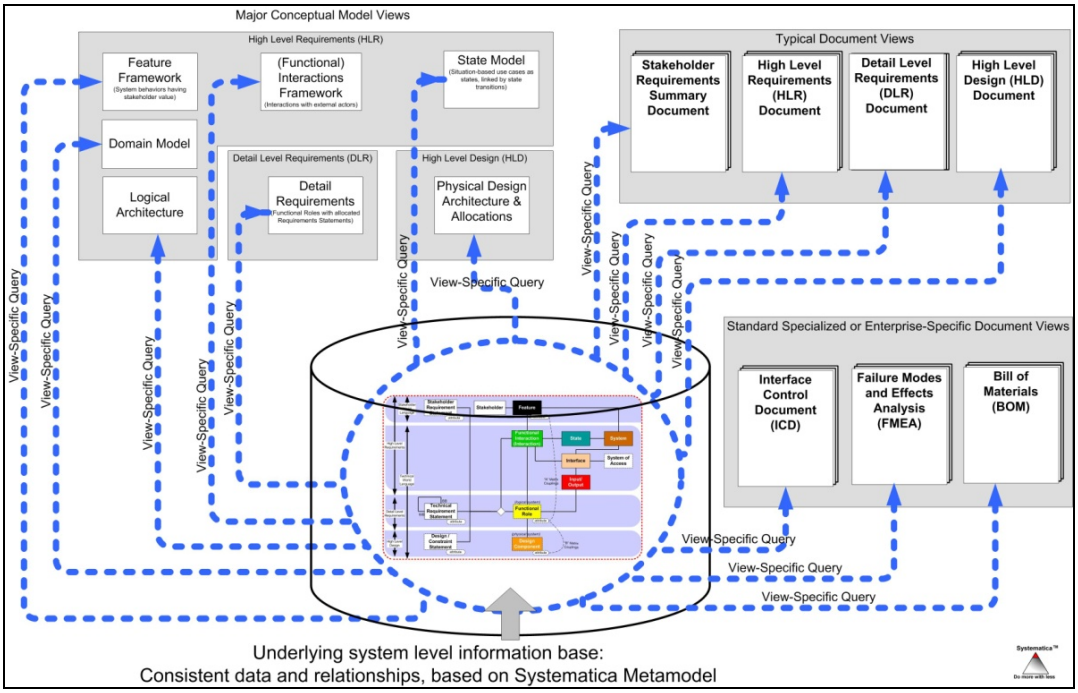

Figure 8: Generation of (Redundant) Views from a Non-redundant Database

The S* Model goes farther, by pointing out redundancies not always recognized; e.g.:

- FMEA Functional Failures vs. Requirements (Counter-Requirements) [Schindel 2010]
- FMEA Failure Effects vs. Stakeholder Features (noted earlier above)
- ICDs vs. System Requirements
- CONOPS and Use Cases vs. System Requirements, Features

Such "redundancies" are really deep insights that make model construction easier & reinforcing: We can still produce all these views, but with less effort and greater consistency.

Measures of model complexity. Models communicate information, as quantified in communication theory [Shannon 1963]. More recently, complexity of objects has been quantified in algorithmic information theory (AIT or Kolmogorov complexity) using the "smallest program" capable of constructing the object or its behavior [Li and Vitany 1997, Chaiten 2005]. The minimality of S* models (measured in bits) also has several practical sides:

- Clarifying "too small" versus "big enough" models: The S* metamodel reminds us of types of systems engineering information that, if omitted, will leave us with an incomplete description of a subject system's requirements, design, or connecting relationships. A practical example is the use of states in a requirements model, reminding us that for any requirement statement, "when does this requirement apply?" is a fair (and often not explicitly answered) question. We may omit this information for pragmatic reasons, but are reminded of what we have not communicated.
- Reducing redundancy and associated inconsistency: Although documents or other task-oriented views generated from an S* model may be redundant, the information in an S* model is not. The consistency of a large number of redundant derived documents and views is easier to maintain or check against a single minimal model.

So, how big? How does an S*model-based compare in size to a traditional systems engineering prose-based description? A practical discovery is that a typical S* model of technical requirements is more complete than a corresponding traditional technical requirements document. Being more complete, it is bigger, not smaller! Figure 9 illustrates some typical sizes. Keep in mind the original question was: What information is essential?

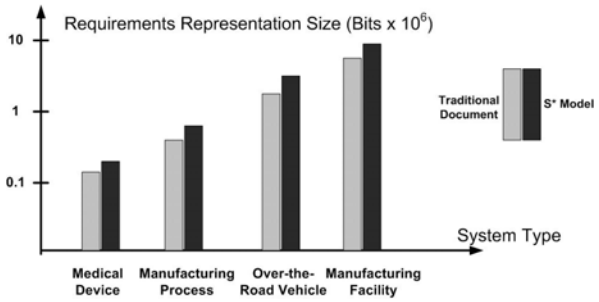

Figure 9: Typical sizes for models and traditional systems engineering documents

Using Patterns to Compress Models

The "starting from scratch" SE process delusion. One of the most significant causes of perceived complexity of the systems engineering process is the fact that most descriptions of the process seem to (implicitly) involve an assumption (judging from the steps they describe)

that is nearly always false for real projects--that the project is "starting from scratch" in a "clean sheet" engineering project on a system for which there are no significant historical precedents. Accordingly, the process systematically seeks out the needed information and processes it into a form usable by the project [ISO/IEC 15288, INCOSE SE Handbook].

On the contrary, real projects are most often concerned with engineering similar (but different) systems across different product generations, applications, configurations, or market segments. At the very least, we are typically engineering whiz-bang product X as the latest improvement in a long line of previous products in the same domain—but with some new differences, big or small. In some cases, we are even planning a product line of related products as a whole:

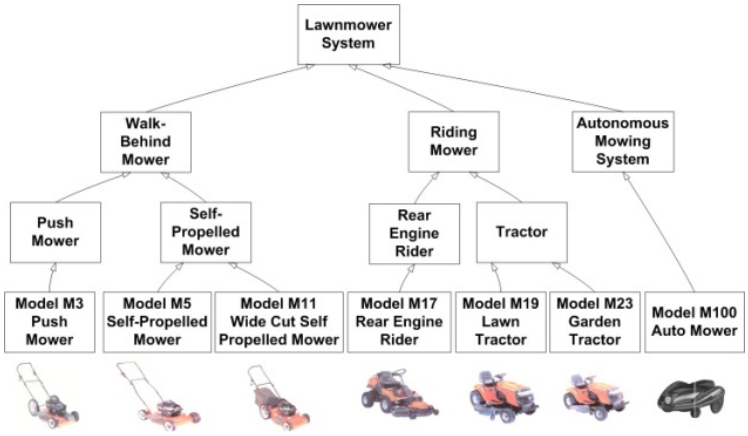

Figure 10: Families of Systems—Whether Generations or Product Lines

In spite of this reality, very little of the descriptions of the SE process is typically about *more efficiently leveraging what we already know* about the target systems. Typically these descriptions make some mention of consulting documents or lessons learned about similar projects, but very rarely is there a procedural discipline *focused specifically on engineering of what we could call "variable sameness*". Something more than a database of useful past requirements, or cloning the last project document from the engineer's desk drawer (a dominant paradigm) is suggested here—an equivalent to perturbation theory in mathematics.

Pattern-Based Systems Engineering (PBSE). Over several decades, we have developed and practiced what we call Pattern-Based Systems Engineering (PBSE) across a range of domains, including carrier grade telecommunications, engines and power systems, automotive and off road heavy equipment, telecommunications, military and aerospace, medical devices, pharmaceutical manufacturing, consumer products, and advanced manufacturing systems [Schindel and Smith, 2002; Schindel, 2005b, Bradley et al 2010]. Engineers in all of these and many other domains spend most of their company's engineering resources developing or supporting systems that virtually always include major content from repeating system paradigms at the heart of their business (e.g., core ideas about airplanes, engines, switching systems, etc.). In spite of this, the main paradigm apparent in most enterprises to leverage "what we know" is to build and maintain a staff of experienced technologists, designers, application engineers, or other human repositories of knowledge. There is typically little evidence of a "Maxwell's Equations" of first principle-based discipline of "variable sameness" in the engineering of these systems.

Although engineering "patterns" already have precedent in systems and software engineering (Gamma, 1995; Alexander, 1977; Haskins, 2005; Cloutier and Verma, 2007), these are often

relatively informal approaches to capturing and re-applying certain general ideas, supporting by templates of one sort or another. By contrast, in PBSE what we are doing is to *extend MBSE* through the use of *formally configurable and re-usable SE models*. Specifically, an S* Pattern is a re-usable, configurable S* Model of a family (product line, set, ensemble) of systems:

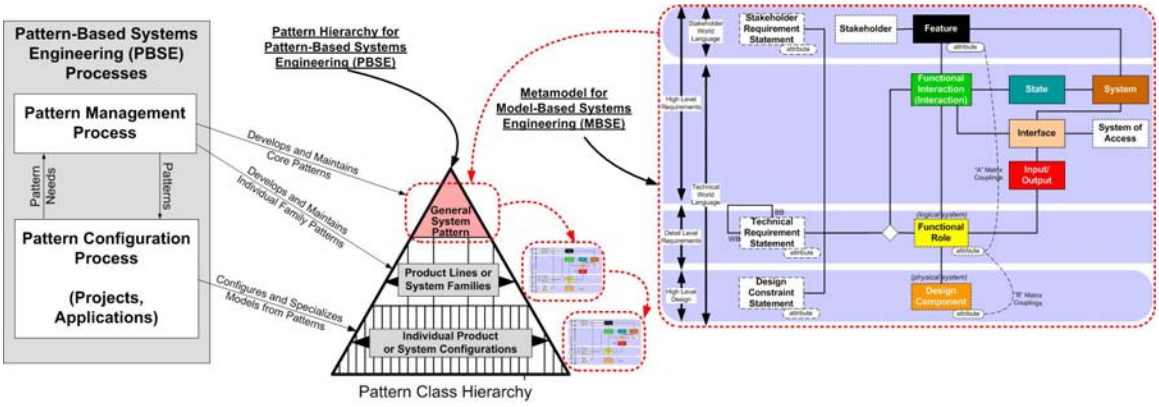

Figure 11: An S* Pattern Is A Configurable, Re-usable S* Model

Pattern Configurations. Such Patterns are ready to be configured to serve as Models of individual systems in projects. "Configured" here is specifically limited to mean that pattern model components are populated / de-populated, and that pattern model attribute (parameter) values are set--both based on configuration rules that are part of the Pattern. Patterns are based on the same Metamodel as "ordinary" Models.

Because of this disciplined approach to "configuration" as a limited case of specialization, relatively dramatic simplifications can frequently occur in the typical engineering process. A table of configurations illustrates how patterns facilitate compression. The rows of the table represent aspects of the model such as Stakeholder Features and their Attributes, Functional Roles, Requirements Attributes, Design Components, Interfaces, etc.,

Table 2: Pattern Configurations Table

Lawnmower Product Line: Configurations Table									
		Units	Walk-Behind Push Mower	Walk-Behind Mower Self-Propelled	Walk-Behind Self-Propelled Wide Cut	Riding Rider	Riding Tractor	Riding Mower Tractor	Autonomous Autonomous
			Push Mower	Self-Propelled	Wide Cut	Rider	Lawn	Garden	Auto Mower
	Model Number		M3	M5	M11	M17	M19	M23	M100
	Market Segment		Sm Resident	Med Resident	Med Resident	Lg Resident	Lg Resident	Home Garden	High End Suburban
Power	Engine Manufacturer		B&S	B&S	Tecumseh	Tecumseh	Kohler	Kohler	Elektroset
	Horsepower	HP	5	6.5	13	16	18.5	22	0.5
Production	Cutting Width	Inches	17	19	36	36	42	48	16
	Maximum Mowing Speed	MPH	3	3	4	8	10	12	2.5
	Maximum Mowing Productivity	Acres/Hr			1.6				
	Turning Radius	Inches	0	0	0	0	126	165	0
	Fuel Tank Capacity	Hours	1.5	1.7	2.5	2.8	3.2	3.5	2
	Towing Feature						x	x	
	Electric Starter Feature				x	x	x	x	
	Basic Mowing Feature Group		x	x	x	x	x	x	x
Mower	No. of Anti-Scalping Rollers		0	0	1	2	4	6	0
	Cutting Height Minimum	Inches	1	1.5	1.5	1.5	1	1.5	1.2
	Cutting Height Maximum	Inches	4	5	5	6	8	10	3.8
	Operator Riding Feature					x	x	x	
	Grass Bagging Feature		Optional	Optional	Optional	Optional	Optional	Optional	
	Mulching Feature		Standard	Factory Installed	Dealer Installed				
	Aerator Feature					Optional	Optional	Optional	
	Autonomous Mowing Feature								x
	Dethatching Feature					Optional	Optional	Optional	
Physical	Wheel Base	Inches	18	20	22	40	48	52	16
	Overall Length	Inches	18	20	23	58	56	68	28.3
	Overall Height	Inches	40	42	42	30	32	36	10.3
	Width	Inches	18	20	22	40	48	52	23.6
	Weight	Pounds	120	160	300	680	705	1020	15.6
	Self-Propelled Mowing Feature			x	x	x	x	x	x
	Automatic TransmFeature							x	
Financials	Retail Price	Dollars	360	460	1800	3300	6100	9990	1799
	Manufacturer Cost	Dollars	120	140	550	950	1800	3500	310
Maintenance	Warranty	Months	12	12	18	24	24	24	12
	Product Service Life	Hours	500	500	600	1100	1350	1500	300
	Time Between Service	Hours	100	100	150	200	200	250	100
Safety	Spark Arrest Feature		x	x	x	x	x	x	

A different way to organize SE processes: PBSE offers us a different (and potentially simpler) way to view the organization of the SE process areas. Instead of dividing, them by their ISO 15288 type functionality first, we can divide them into two major processes (see Figure 11):

- Pattern Management Process: Generates the underlying family model, and periodically updates it based on application project discovery and learning
- Pattern Configuration Process: Configures the pattern into a specific model for application in a project

The second of these two processes may well contain what could be viewed as outcome equivalents to the ISO 15288 process areas, but they can be viewed in a much different light if they are first each asking how to produce their products from what is already known (the Patterns that govern the target system – not the engineering process). Much of the more complex formal machinery of systems engineering can then be "hidden" in the other process—the Pattern Management Process, in which a much smaller number of people's efforts are leveraged by a larger population in the second process. In this approach, Patterns become valued IP, and are sometimes even financially capitalized as a form of "software".

As a start toward a "thermodynamics of patterns", the Gestalt Rules [Schindel, 1997] describe what it means for a holistic system model to either conform to or not conform to a more general holistic system model. For example, if we develop state models of aircraft over mission profiles that include preparation, take-off, climb, cruise, combat, return, landing, etc., then how can we compare fixed-wing, helicopter, VTOL, civil, and other aircraft?

Compression of models, using patterns. Each column in the table is a compressed system representation with respect to ("modulo") the pattern. The compression is typically very large. The compression ratio tells us how much of the pattern is variable and how much fixed, across the family of potential configurations. Refer to Figure 12

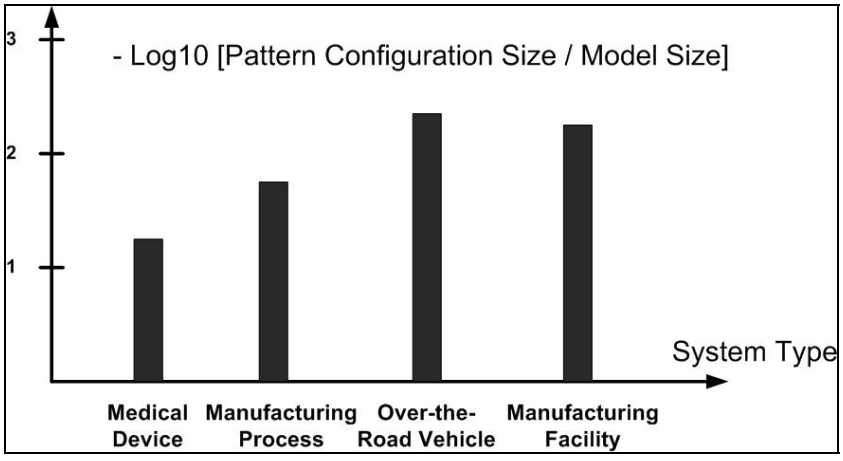

Figure 12: Pattern Compression

Connection to Minimum Description Length (MDL) theory. In MDL and Kolmogorov Complexity theories applied to complexity, there is an idea of the representation of a system "modulo" a certain language used to describe it. Likewise in PBSE, the configuration of a pattern is a "description" of that system within the space of systems governed by that pattern. If we assume that the pattern itself is already known or accepted, then the configuration

information becomes a (much shorter) description of "where in the pattern space the particular configuration is", tying down the degrees of freedom offered by the pattern.

If the language that emerges from a pattern is extremely flexible (e.g., English prose), then the degrees of freedom are very large indeed, and the configuration data itself must be extensive. But, if the pattern is based on a construct like the S* Metamodel, then the domain-specific SE language that emerges from that pattern is orders of magnitude more restrictive, and the configuration information is accordingly much simpler and easier to understand, analyze, and communicate.

All Models Are Configurations of More Abstract Patterns

We arrive at a core idea for simplifying the SE process. Instead of asking how to adopt all the sophisticated machinery of formal PBSE, we can alternatively realize that all models are configurations of more abstract patterns, whether we formalize those patterns or not. Moreover, even non-MBSE engineering projects are in fact creating informal "configurations" of informal "patterns" every day, and have been all along. As evidence of this, consider all the "important known stuff" that we don't always write down in projects--the content of industry and enterprise standards comes first to mind. We "invoke" these by reference, but we rarely import explicitly all of their content into our specifications. They become stacks of additional "side" documents that vex designers, suppliers, and others who must conform to them or verify conformance.

What is missing in (most but not all of) these traditional approaches is a sufficient machinery to truly configure these patterns of "external" data for a given project. At best, we might typically see citations of particular sections of these documents that are chosen to apply. More typically, we are left to wonder which parts of these stacks may apply and which do not. By adopting some of the simplest elements of PBSE discipline, once onerous processes can become assets, as we move more rapidly with configuration data, supported by less frequently consulted (but nevertheless available when needed) "pattern" information in these other references.

Conclusions

1. The specific MBSE and PBSE methods discussed here have been successfully applied across a wide range of domains: Transportation, Mil/Aero, Communications, Medicine/Healthcare, Advanced Manufacturing, Consumer Products.
2. The minimum base of information required to perform specific SE process areas is greatly clarified by MBSE metamodel understanding.
3. Minimal MBSE models contain information missing from many projects, causing practical project problems.
4. Minimal underlying models generate the redundancies needed across different task-based artifacts, with greater consistency or less effort to maintain that consistency.
5. Formalization of Patterns as configurable Models leads to further size compression: Configurations.
6. All models are actually configurations of more abstract patterns. Realizing and exploiting this can turn the previous "deadweight" of standards and other external references into powerful assets for accelerating work.

References

1. Ahmed, J., Hansen, J., Kline, W., Peffers, S., Schindel, W. 2011. All innovation is innovation of systems: An integrated 3-D model of innovation competency. To appear in *Proceedings of the 2011 American Society for Engineering Education Annual Conference*, Vancouver, BC.
2. Alexander, Christopher; Sara Ishikawa, Ingrid Fiksdahl-King, Shlomo Angel. 1977. *A pattern language: Towns, buildings, construction.* New York: Oxford U. Press.
3. Ashby, W. Ross. 1957. *An introduction to cybernetics*. London: Chapman & Hall.
4. Bar-Yam, Y. 2003b. When systems engineering fails—toward complex systems engineering. *Proceedings of the International Conference on Systems, Man & Cybernetics*, Vol 2, 2021-2028. Piscataway, NJ: IEEE Press.
5. ———. 2005. About engineering complex systems: multiscale analysis and evolutionary engineering. ESOA 2004, LNCS 3464, pp 16-31, Spinger-Verlag, 2005.
6. Bradley, J, Hughes, M, Schindel, W. 2010. Optimizing delivery of global pharmaceutical packaging solutions, using systems engineering patterns. *Proceedings of the INCOSE 2010 Symposium.*
7. Braha, D., A. Minai, Yaneer Bar-Yam, eds. 2006. *Complex engineered systems: Science meets technology,* City: Springer.
8. Chaitin, Gregory. 2005. *Metamath: The quest for omega,* New York: Pantheon, 2005.
9. Cloutier, Robert J., Dinesh Verma. 2007. Applying the concepts of patterns to systems architecture. *Systems Engineering.* Wiley. Vol 10, No. 2. pp 138-154.
10. Duda, Richard. O., Peter E. Hart, David G. Stork. 2001. *Pattern classification,* (2nd ed.), New York: Wiley.
11. Estafan, J. 2008. Survey of model-based systems engineering (MBSE) methodologies. INCOSE MBSE Initiative.
12. Gamma, E., R. Helm, Ralph Johnson, J. Vlissides. 1995. *Design patterns: Elements of reusable object-oriented software*. Reading, MA: Addison-Wesley.
13. Gould, S. J. 2003. *The hedgehog, the fox, and the magister's pox: Mending the gap between science and the humanities*. New York: Three Rivers Press.
14. Grunwald, P. 2007. The minimum description length principle. Cambridge, MA: MIT Press.
15. Gunyon, R., and Schindel, W. 2010. Engineering global pharmaceutical manufacturing systems in the new environment. *Proceedings of the INCOSE 2010 Symposium.*
16. Haskins, Cecilia. 2005. Application of patterns and pattern languages to systems engineering. Paper presented at the 15th annual international symposium of the international council on systems engineering, Rochester, NY.
17. Haskins, Cecilia, ed. 2010. *INCOSE systems engineering handbook, Version 3.2.* Seattle, WA: International Council on Systems Engineering.
18. INCOSE HSIG web site. http://www.incose.org/practice/techactivities/wg/hsi/
19. INCOSE MBSE web site : http://www.incose.org/practice/techactivities/modelingtools/mdsdwg.aspx.
20. INCOSE SSWG web site: http://www.incose.org/practice/techactivities/wg/syssciwg/
21. ISO 10303 AP233 web site. http://www.ap233.org/
22. ISO/IEC 15288: 2002. Systems engineering – System life cycle processes. Geneva: International Organization for Standardization.
23. Karayanakis, N., *Computer-assisted simulation of dynamic systems with block diagram languages.* CRC Press, 1993.
24. Kauffman, Stuart. 2000. *Investigations* New York: Oxford University Press.

25. Kuras, M. L., B. E. White. 2005. Engineering enterprises using complex-system engineering. Paper presented at the annual international symposium of the International Council on Systems Engineering, July, Rochester, NY.
26. Li, Ming, Vitany, Paul. 1997. *An introduction to Kolmogorov complexity and its applications*. Second edition. Springer.
27. Mellor, Stephen; Marc J. Balcer. 2002. *Executable UML: A foundation for model-driven architecture.* Boston: Addison-Wesley.
28. Schindel, W. 1996. Systems engineering: An overview of complexity's impact. Tech Paper 962177, SAE International.
29. _____. 1997. The tower of Babel: Language and meaning in system engineering. Technical Report No. 973217 SAE International.
30. ———. 2005a. Requirements statements are transfer functions: An insight from model-based systems engineering. Paper presented at the annual international symposium of the International Council on Systems Engineering, July, Rochester, NY.
31. ———. 2005b. Pattern-based systems engineering: An extension of model-based systems engineering. INCOSE TIES tutorial presented at 2005 INCOSE Symposium.
32. _____. 2006. Feelings and physics: Emotional, psychological, and other soft human requirements, by model-based systems engineering. *Proceedings of the INCOSE 2006 International Symposium.*
33. _____. 2010. Failure analysis: Insights from model-based systems engineering. *Proceedings of the INCOSE 2010 International Symposium.*
34. _____. 2011. Systems engineering for advanced manufacturing: Unit op insights from model-based methods. To appear in *Proceedings of the INCOSE 2011 International Symposium.*
35. Schindel, William D., Vern R. Smith. 2002. Results of applying a families-of-systems approach to systems engineering of product line families. Technical Report 2002-01-3086. SAE International.
36. Shannon, Claude. 1963. *A mathematical theory of communication*. Champaign, IL: University of Illinois Press.
37. Snow, C.P. 1960. *The two cultures.* Cambridge: University Press. pp. 181. ISBN 978-0521457309 (second edition; 1993 reissue).
38. SysML Partners web site. http://www.sysml.org/

Biography

William D. Schindel is president of ICTT System Sciences, a systems engineering company, and developer of the Systematica™ Methodology for model and pattern-based systems engineering. His 40-year engineering career began in mil/aero systems with IBM Federal Systems, Owego, NY, included service as a faculty member of Rose-Hulman Institute of Technology, and founding of three commercial systems-based enterprises. He has consulted on improvement of engineering processes within automotive, medical/health care, manufacturing, telecommunications, aerospace, and consumer products businesses. Schindel earned the BS and MS in Mathematics, and was awarded the Hon. D.Eng by Rose-Hulman Institute of Technology for his systems engineering work.