INCOSE 2005 Symposium "Best Paper" Award in Modeling and Tools

Requirements Statements Are Transfer Functions: An Insight from Model-Based Systems Engineering

William D. Schindel
ICTT, Inc., and System Sciences, LLC
100 East Campus Drive, Terre Haute, IN 47802
812-232-2062 schindel@ictt.com

Abstract. Traditional systems engineering pays attention to careful composition of prose requirements statements. Even so, prose appears less than what is needed to advance the art of systems engineering into a theoretically-based engineering discipline comparable to Electrical, Mechanical, or Chemical Engineering. Ask three people to read a set of prose requirements statements, and a universal experience is that there will be three different impressions of their meaning. The rise of Model-Based Systems Engineering might suggest the demise of prose requirements, but we argue otherwise. This paper shows how prose requirements can be productively embedded in and a valued formal part of requirements models. This leads to the practice-impacting insight that requirements statements can be non-linear extensions of linear transfer functions, shows how their ambiguity can be further reduced using ordinary language, how their completeness or overlap more easily audited, and how they can be "understood" more completely by engineering tools.

Systems Engineering Prose

Traditional Requirements Discipline. Composing good requirements statements prose has a long tradition in systems engineering. As described in (Buede 2000), systems engineers are typically instructed that effective requirements statements should be:

- Unambiguous
- Understandable
- Correct
- Concise
- Traceable, Traced
- Design Independent
- Verifiable
- Unique
- Complete
- Consistent
- Comparable
- Modifiable
- Attainable

The resulting requirements describe systems, are stored in databases, expressed in requirements documents, and interpreted by people. (See Figure 1.)

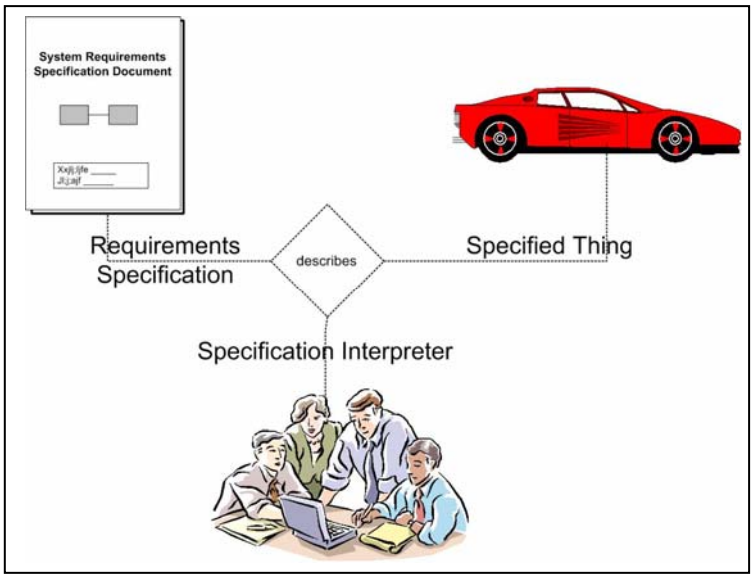

Figure 1: Specification Documents Describe Things to Interpreters

Growing Challenges to Prose. Well-structured prose requirements statements are of course more effective for this worthwhile and practical care. But, are traditional prose requirements compositional principles effective enough for the future demands of systems engineering, as it strives to move from a craft-like body of knowledge to a scientifically-founded engineering discipline, comparable to electrical, mechanical, or chemical engineering? Engineered target systems are becoming more complex, more mission-critical, more risk-averse, more in need of clear human understanding, demanding of faster development cycles, and supported by larger engineering teams, who are interoperating with more engineering tools. Is it reasonable to expect that well-composed prose will be up to these rising technical demands?

Is good prose the bulwark of good engineering, or of good literature? If Newton and his followers had been forced to use only prose to express their reasoning, would today's engineers be using orbital mechanics to design mission systems? The *Principia* (Newton 1668) straddles the transition from prose-based geometric proof to the power of mathematics. Should we expect that our systems engineering prose will be replaced with mathematical equations?

Systems Engineering Models—Replacement or Extension?

Model-Based Systems Engineering. INCOSE has helped to lead progress to model-based methods for systems engineering (INCOSE MBSE 2004). The use of graphical and other forms of models has appeared in systems engineering through application of graphically-focused modelling languages (backed by underlying information metamodels) such as IDEF0, UML™, and most recently SysML™ modelling languages, described in (Buede 2000, Oliver 1999, Booch et al 1999, SysML 2004, AP233 2004).

Models are data structures (often, but not always, represented graphically) intended to *explicitly* represent facts (e.g., requirements, designs) about systems (refer to Figure 2). These facts might otherwise be *implicit* in knowledge of experienced systems engineers or domain experts. Without explicit models, these facts are not equally obvious to all, require reading "between the lines", and are opportunities for mis-understandings, engineering process errors, rework, or failures.

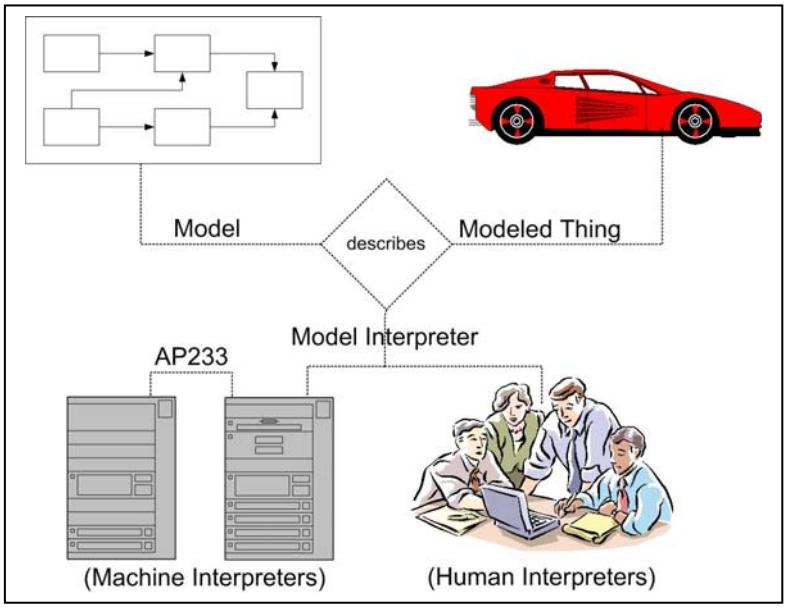

Figure 2: Models Describe Things to Interpreters

The model-based engineering approach, not fully reviewed here, intends that these models, when compared to earlier approaches:

- Are more explicit
- Are more compact
- Enhance visualization, understanding, and communication
- Enhance the formal underlying theoretical structure of engineering information, to improve ability to analyze, simulate, (execute the model), and even synthesize
- Enable database-driven processes instead of document-driven processes
- Improve shared "understanding" of the meaning of models to be exchanged between machines (engineering tools) and humans
- Provide theoretical foundation that was not available in earlier prose-based approaches, supplementing intuition with discipline

Electrical schematic diagrams and mechanical drawings, if drawn according to learned disciplines, provide engineers with relatively unambiguous models of systems, compared to the typically more ambiguous experience with prose-based systems engineering requirements documents (Glasgow et al 1995). Not all diagrams are unambiguous, as famously illustrated by (Escher 1992). How can general system requirements be expressed as clearly and unambiguously as the design communicated by an electrical schematic diagram?

Will Diagrams Replace Prose? Progress occurring with model-based systems engineering might lead to the expectation that the graphic components of models will eventually replace the use of prose-based requirements statements altogether. We argue differently here, in spite of the claim that "a picture is worth a thousand words".

Our experience is that the most productive outcome is not the total replacement of prose with diagrams, but a merging of these two forms of information, into a total formal model that includes both. Current efforts to incorporate prose into models in some fashion are described in (SysML Partners 2004) for SysML modelling and in (AP233 2004) for the related AP233 activity. We will describe the embedding of prose into the model, as a first class part of that model. The approach we will describe is something more than just embedding requirements prose as unstructured text. Our inspiration for how prose should be embedded in models comes from examining the underlying meaning of the original requirements prose—the special semantics of requirements statements as a specialized subset of all prose.

Given current practices, tools, trends, and standards efforts, this outcome is by no means obvious, and requires both careful theoretical and practical review to understand what is possible and how one might interpret current evolution of practice.

Embedding The Prose In The Models

Transfer Functions. A rich vein in the theory of linear systems is the idea of *transfer functions,* as described in (Churchill, 1958). Transfer functions describe the relationship between system inputs and outputs, typically in the frequency domain:

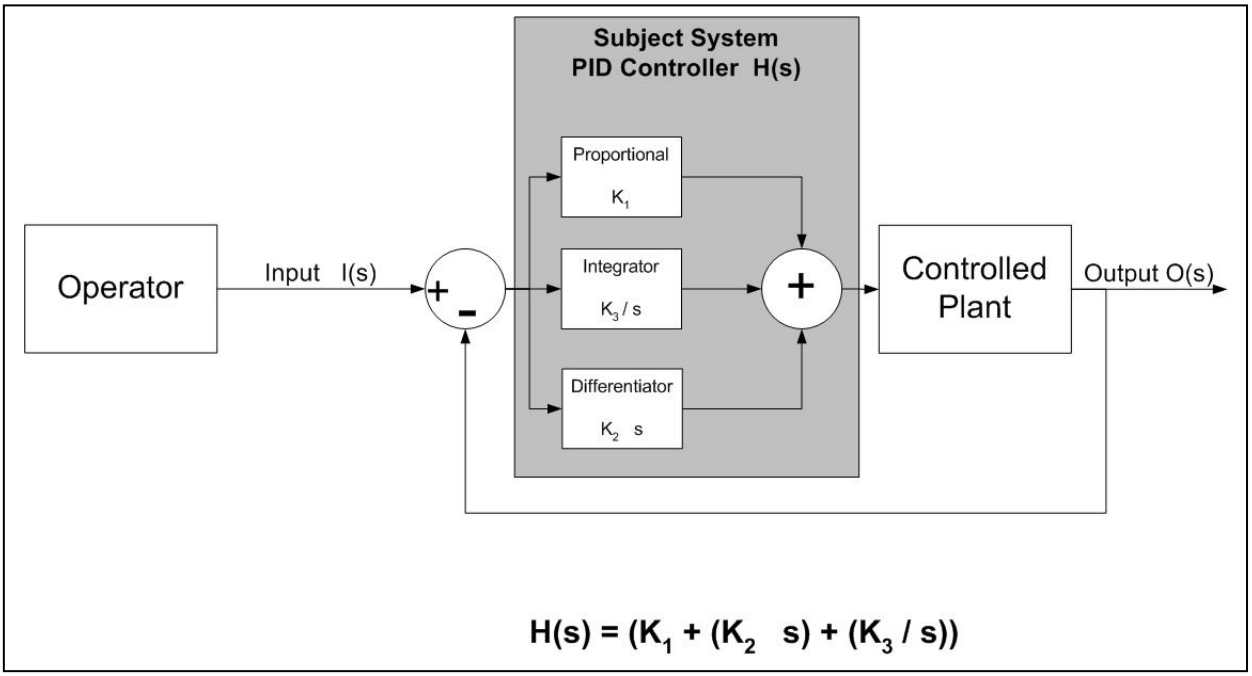

Figure 3: Logical Architecture of a PID Controller

Such a transfer function, parameterized by attributes (K1, K2, K3), completely characterizes the behavior of the associated linear system—all of its behavioral characteristics (whether in the frequency or time domain) can in principle be derived from the transfer function description. Unfortunately this utility appears limited, as most aspects of the systems encountered by engineers are non-linear, and therefore not described in this way.

Nevertheless, we retain one idea from linear systems and their transfer functions—the benefit of characterizing the external behavior of a system in the form of *relationships between its inputs and outputs*. Engineers recognize the value of somehow characterizing a system's externally visible behavior, whether in the form of data sheets, commercial specifications, or otherwise. But can we expect to do this in the form of algebraic or differential equations? Most (perhaps all) engineered systems evade practical description in the form of such equations—nonlinear or otherwise. Furthermore, system stakeholders often don't speak the language of mathematics.

What we are interested in retaining (and abstracting) is the idea of *statements of relationships* between inputs and outputs, whether in the form of simple mathematical equations or something else. The idea is to describe the relationship of values of the system's outputs to the values of its inputs-- including the impact of time history. Why is characterizing such behavior so important?

What are Functional Requirements, Really? Traditional systems engineering teaches us that requirements are to describe *what* a system should do, not *how* it does it. Just what does this really mean, in the language of science, engineering, and mathematics? Whether we use prose requirements statements or some other form of description, the need is to describe the system as a "black box"—describing its required behavior as "seen" by the other systems with which it interacts (Figure 4), without discussing its internal design implementation. This certainly sounds like a formal characterization of the system, and could be described in terms of relationships between its inputs and output.

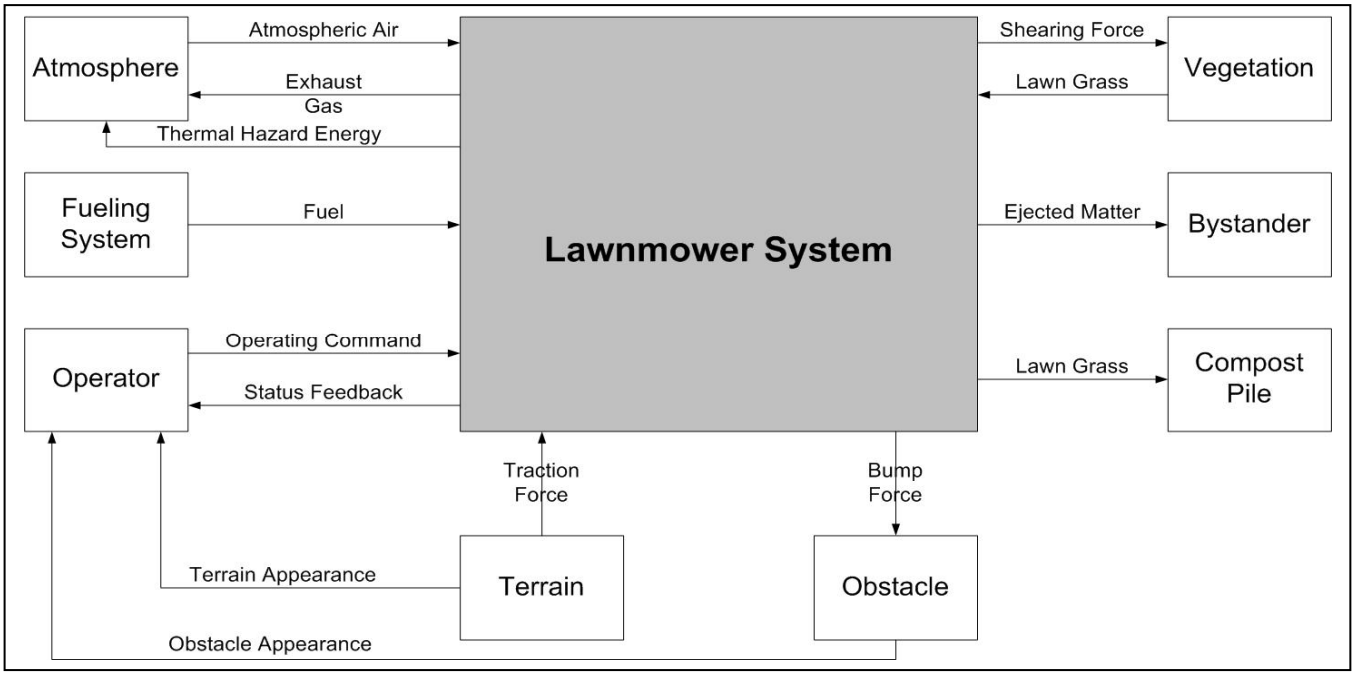

Figure 4: System External Inputs and Outputs

What About "Non-Functional" Requirements? Why don't we typically think of "requirements" as this kind of formal external characterization? At least two issues usually suggest that "requirements" for a system might not be exactly the same idea as this formal behavioral characterization of the system:

1. Stakeholder requirements in the non-technical language and perspective of stakeholders may seem different than such technical system characterization—witness, for example Quality Function Deployment's (QFD's) approach, as described in (Clausing et al 1988).
2. Requirements for system reliability, manufacturability, supportability, or the other "-ilities", as well as system capacities, tolerances, or other requirement categories may seem different than system technical characterization—to the point that these are sometimes referred to as "non-functional" requirements, as in (Chung et al 1999).

However (as discussed later below), all these different needs eventually *map to and can be productively expressed as* the external behavior of the system, expressed as the totality of its input-output relationship characteristics. Indeed, for all of these, systems engineers typically "derive" technical language requirements that are in principle objectively testable (or otherwise analyzed) input-output behaviors at the interfaces of the system, while recognizing that such "technical requirements" are different (transformed from) the original stakeholder requirements.

The theme of our argument is that all requirements statements are input-output relationships, or generalizations of transfer functions, expressing external behavior. (Design constraints are not a part of this argument, but are likewise traceable to desired external behaviors, which should be included.) A key attraction of this view is to take better advantage of the edifice of the scientific-mathematical understanding based on physical interactions that was built by giants such as Newton, Hamilton, Maxwell, and others (Newton 1668), (Hamilton 1834), (Sussman 2001), (Landau et al 1976), as the scientific underpinnings of modern engineering disciplines. Indeed, the overall system engineering methodology from which this view is taken (Schindel 2002; Schindel et al 2002) is based upon the concept of interactions of multiple components, versus the behavior of any single component in isolation. While this viewpoint is foundational in science, systems engineering of requirements sometimes takes the isolated system perspective, leading to later integration challenges.

We treat all the requirements on a system as ultimately expressed (possibly through derivation mappings from stakeholder needs or other forms) in the form of system interaction behavior at external system boundaries. This behavior can be described in terms of relationships between system inputs and outputs that characterize the system. These relationships are frequently parameterized by system attributes. This also allows us to take advantage of both contemporary modeling languages (SysML Partners 2004; AP233 2004), as well as the success of physics.

The Return of Prose to the Model. The approach described here, then, is to see requirements statements prose as existing solely for the purpose of expressing required system input-output behavioral relationships. This is not the most widely held or traditional view of requirements statements. We can conceive of our typical prose sentences as a kind of generalization of mathematical equations, expressing what in the end really are relationships between input and outputs. Indeed, this approach is familiar in the world of VHDL characterization of digital electronics (Ashenden 1996) or propositional calculus assertions about system logical behavior (Carnap 1958). It also fits the cases in which there are algebraic, differential, or integral equations relating inputs and outputs mathematically (whether deterministically or stochastically), as well as fuzzy relationships (Zadeh 1965). Similarly, requirements statements may describe relationships by the use of tables, graphs, or other representations of I/O relationships. Developers of modeling languages benefit from noting that equations, graphs,

tables, may not need improvement, but instead direct incorporation into the model. But what is the practical implication for regular prose sentences in English or other national languages?

The first implication of this approach is a simplified way to think about requirements statements: As shown in Figure 5, requirements statements for a given subject system need only contain:

1. References to inputs and outputs
2. Statements of relationships between them, including attributes that parameterize those relationships

This simplifies the process of composing, as well as interpreting, analyzing, and auditing, these requirements statements. It does so by more than traditional prose grammatical means—it does so by taking a mathematical – physical modeling view of systems, in the tradition of engineering and science. At the same time, it preserves the "normal everyday language" aspects of requirements statements.

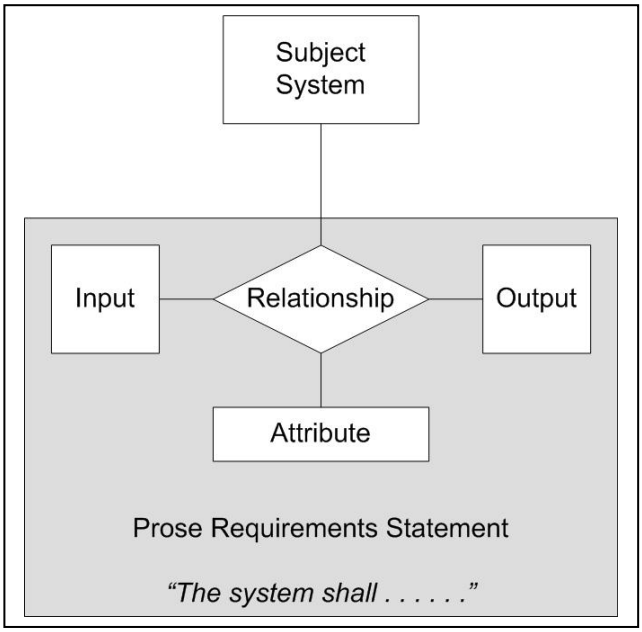

Figure 5: Prose Requirements Metamodel

Figure 5 is an information model of prose requirements statements, and some such statements will contain multiple (or no) instances of the classes it describes. While not every requirements statement needs to contain inputs, outputs, or attributes, every such statement should contain at least one of these, and will refer to at least one relationship. There should always be a clear association to a subject system. The complete characterization of the total relationship between system inputs and outputs is the union of all that system's prose requirements statement models.

In the following example requirements statements, the prose has been punctuated to make the components of Figure 5 more evident:

- Inputs and outputs are underlined.
- [Attributes] are in brackets.
- Relationships are italicized.

1. "The Lawnmower System shall *operate* with [Hourly Mowing Capacity] of at least 1 level ground acre per hour, at [Max Elevation] up to 5,000 feet above sea level, and [Max Ambient Temperature] of up to 85 degrees F., at up to 50% [Max Relative Humidity], for [Foliage Cutting Capacity] of Acme American Standard one week Lawn Grass."
2. "The Lawnmower System shall *operate* using Fuel consisting of gasoline having a [Min Octane Rating] of not less than 92, combusted with Atmospheric Air."
3. "The Lawnmower System shall *operate* with [Fuel Economy] of at least 1 hour / gallon at [Min Elevation] of 0 feet ASL, at [Max Ambient Temperature] 85 degrees F., 50% [Max Relative Humidity], for Acme American Standard one week Lawn Grass."
4. "The Lawnmower System shall *operate* with [Elevation Derating] of 10% improvement in [Fuel] per 1,000 feet of elevation reduction, to a [Min Elevation] of 0 feet ASL.
5. "The Lawnmower System shall *operate* meeting the more demanding of state and federal standards for [Max Gaseous Pollution] and [Max Particulate Pollution] of Exhaust Gas.
6. "The Lawnmower System shall *operate* with [Operating MTBF] no less than 500 hours."
7. "The Lawnmower System shall *operate* so as to protect the Operator from Thermal Hazard Energy by maintaining all accessible metallic surfaces at a [Maximum Surface Temperature] of less than 180 degrees F."

Decomposition, Logical Architecture, and Allocation. Prose requirements statements are traditionally transformed into derived statements that describe requirements having smaller scope and/or greater specificity or detail. This traditional approach is matched in the perspective described here by the process of decomposing (partitioning) a subject system into *logical subsystems*. These are groupings of required external behavior, not design allocations. Refer to Figure 6 for a Logical Architecture partitioning external behavior (not design) of Figure 4.

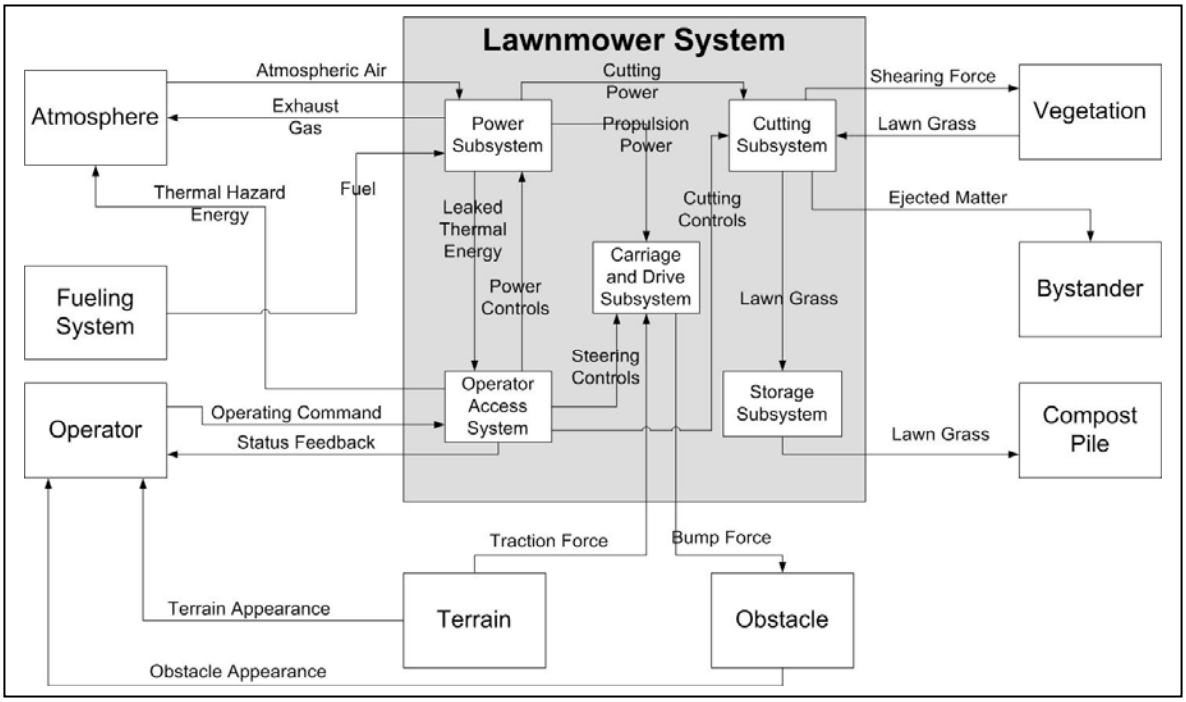

Figure 6: Logical Architecture – Partitioning of Figure 4

In the above example, requirement (1) is decomposed in the same way that Figure 6 decomposes the Lawnmower System into subsystems. Each of the following requirements, derived from requirement (1) above, is allocated to a different logical subsystem of Figure 6:

1.1) "The Power Subsystem shall *generate* [Power Output] of combined Propulsion Power and Cutting Power at [Max Elevation] up to 5,000 feet above sea level, and [Max Ambient Temperature] of up to 85 degrees F., at up to 50% [Max Relative Humidity]."

1.2) "The Carriage and Drive Subsystem shall generate a Traction Force sufficient to *propel* over an [Hourly Mowing Capacity] of at least 1 level ground acre per hour, by converting a Propulsion Power input of [Traction Power Consumption]."

1.3) "The Cutting Subsystem shall generate Shearing Force sufficient to *cut and capture* Lawn Grass of at least 1 ground acre per hour of [Foliage Cutting Capacity] of Acme American Standard one week Lawn Grass, by converting a Cutting Power input of [Cutting Power Consumption]."

This illustrates the introduction of intermediate (internal) input-output variables, as well as subsystem attributes and budgets.

Additional Implications

The above argument leads us to write technical requirements statements (derived from less technical statements of stakeholder needs) that explicitly describe (parameterized) input-output relationships. This has a number of additional implications, including the following areas.

Improved, Inspectable Statement Structure. Expressing requirements in the form described here significantly improves the ability to inspect requirements for completeness, as well as clarity. This is because they are expressed with respect to (and in fact are embedded in and part of) a system model, in which their formal role is now exactly to characterize system outputs with respect to system inputs. We can ask, "Does this set of requirements statements specify the required system output values for all input values (and histories)?" Lack of coverage, as well as overlap or conflict, are more easily detected. A traditional challenge of specifying system requirements is understanding whether we are "done" generating them. While some of this is uncertainty about stakeholder needs, in more complex systems a typical problem is determining whether a candidate requirements specification completely characterizes the system at all.

Implications for Tools. When the micro-structure of requirements statements is understood to be a parameterized statement of relationship between inputs and outputs, then tools can better "understand" the semantics of these statements. Such tools can potentially answer questions about requirements, even to the extent of improving the executability of models in simulation, a goal described in (Mellor 2002) and (Karayanakis 1993). Simulation relationships are directly embedded in the model.

Implications for Models. When the micro-structure of requirements statements is understood to be a parameterized statement of the relationship between inputs and outputs, then the form of

requirements models themselves can be more expressive and explicit than if these statements are simply viewed as strings of textual prose. Refer to the requirements diagrams of (SysML Partners 2004).

Implications for Reuse. By making the key variables of requirements explicit parameters (attributes) of requirements statements, we have improved not only our understanding of key parametric relationships, but also the configurability of those requirements for re-use. Re-usable designs and design platforms arise from the re-usability of requirements, enhanced by this approach. This is discussed further below.

Relational-Symbolic Duality. This approach illustrates the underlying duality of the representation of requirements in symbolic (e.g., prose or equations) versus relational (e.g., graphical or relational models) form, as further discussed in (Schindel 1997), (Hayakawa 1990), (Chomsky and Piaget 1980), (Whorf 1956), and (Marcus 2001).

What Is Left? The Non-Prose Part of Requirements Models

In this paper and the systems engineering methodology it references (Schindel et al 2002) we consider prose requirements statements to be a part of, but not all of, a total model of requirements. The referenced methodology grew out of research seeking the minimal information necessary to describe a system (Schindel 2002). Having clarified above the role of prose requirements statements as one formal part of a total requirements model, it of interest to ask: What is the rest of the requirements model? Why is more needed than the requirements prose alone, if such prose describes all the I/O relationships?

Additional Metamodel Components. While this issue would ultimately take us on a trip through modelling that goes beyond the scope of this paper, it is interesting to briefly see the roles of the rest of a requirements model with respect to that of the prose requirements statements alone. The following additional model components, summarized by Figure 7 and described in (Schindel 2002) are also needed to complete the requirements model, and they answer the listed requirements questions, supplementing the requirements statement prose:

1. Domain Model: What is environment of the subject system? With what external systems (actors) does it interact, through what external interfaces? What are the key relationships of this domain? What external interactions are eligible to be characterized by, and must be covered by, prose requirements statements?
2. Stakeholders and Needs Model: What are the primary stakeholder roles played by people or organizations with a stake in the system, and what are their (voice of the stakeholder) needs?
3. Feature Model: How are the behaviours of the system organized with respect to the values of its stakeholders? What attributes describe these values in stakeholder terms? These are the stakeholder language behaviours eligible and required to be technically characterized by requirements statement prose.
4. State Model: How do the behaviour functional relationships between the subject system and its environment change modalities in the presence of different environmental situations (states)? What is the temporal model of the environment, and when (with respect to that temporal situational model) do different requirements apply?

5. Functional Interaction Model: What is the organization of the technical physical interactions of the system with its environment? How are these interactions described by the input-output relationships described by the prose requirements? What are the key technical attributes describing this behaviour, and how are these coupled to the stakeholder value-based feature attributes?
6. Logical Architecture Model: How is the externally-viewed functional interactive behaviour of the system decomposed and organized by partitioning it into a logical architecture of behaviour? This partitioning describes the decomposition and derivation of lesser scope detailed requirements.

These additional requirements model components provide context and fill out the formal meaning of the requirements statements that are embedded into them. For example, the connection of the requirements statements to the State Model of (4) above illustrates the embedding recently described in (Daniels and Bahill 2004).

Still other components of this metamodel relate these requirements to system design, verification, etc. This overall metamodel is related to, although more abstracted than, the AP233 metamodel described in (AP233 2004).

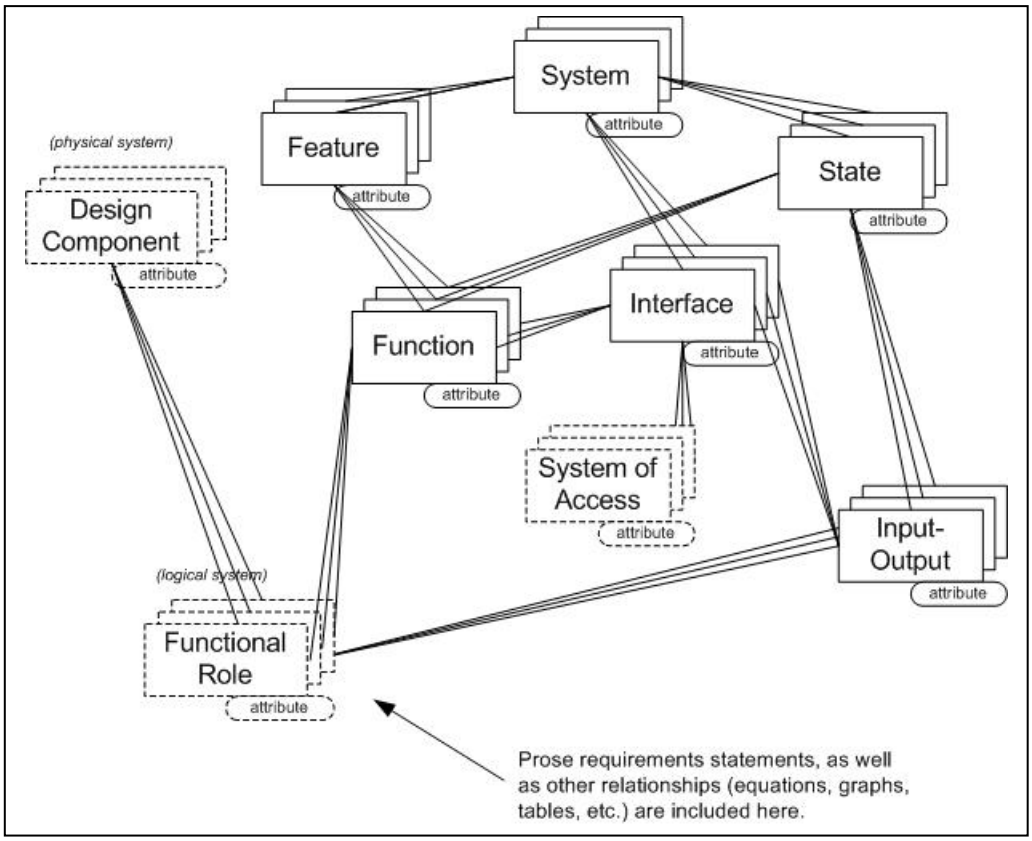

Figure 7: Summary of the Larger Systems Engineering Metamodel

Patterns: Reusable, Configurable Models of Requirements

Reusable designs are possible only because of reusable requirements—some commonality of needs across different market segments, customers, applications, product lines, or sub-systems. *Patterns* are re-usable models. While first popularized in some domains for *design* patterns (Gamma et al 1995; Alexander et al 1977), they are of interest as patterns of *requirements*.

The embedding of parameterized requirements statements in overall requirements models described in this paper, along with the other aspects of the object oriented metamodel of Figure 7, create a modelling framework that enables Pattern-Based Systems Engineering (Schindel, 2002; Schindel and Smith, 2002). This approach introduces patterns as re-usable models, cast in the metamodel of Figure 7.

Using this approach, Figure 8 illustrates the process by which patterns of requirements and designs for generic systems can then be configured or specialized into individual product line families, and ultimately individual product systems. This approach has been applied in a number of Commercial Off the Shelf (COTS) product line enterprises, to enhance COTS portfolio engineering and planning. This approach also facilitates the ongoing expression of organizational learning in the form of updates and refinements to "uncovered" Patterns. A particularly striking benefit of this approach is that it allows large organization practitioners who are less skilled in "clean sheet" original modelling to gain the benefits of model-based engineering. This is accomplished by teaching larger groups the Generic System Pattern models of the enterprise, for their configuration and use. We have found this is more easily learned by larger groups than abstract modelling methodologies.

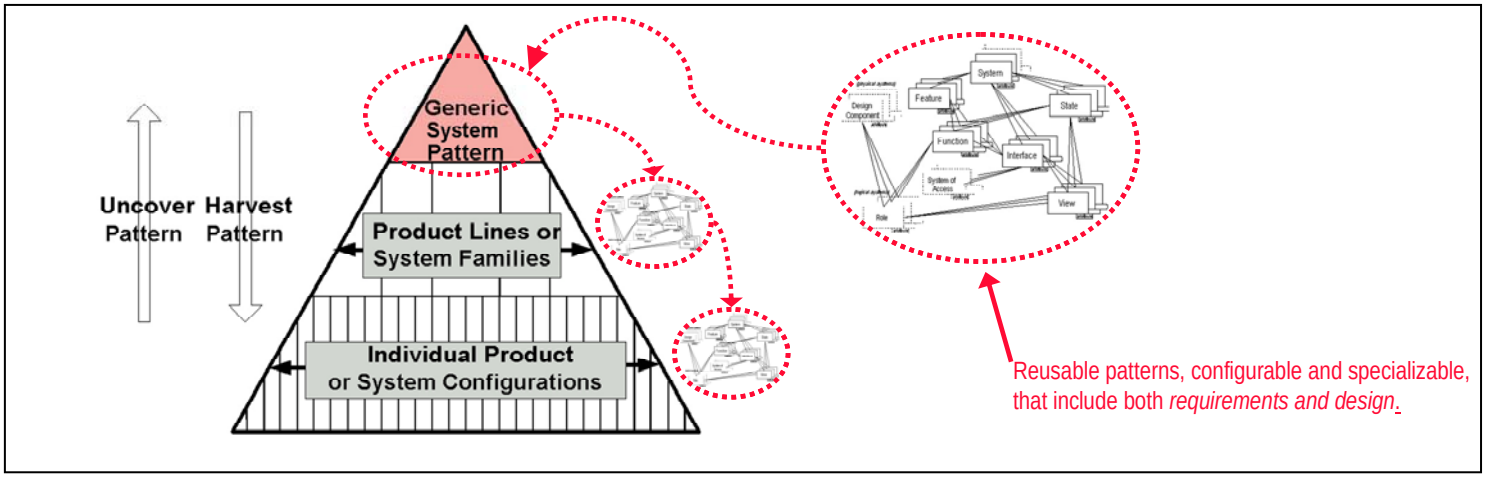

Figure 8: Patterns Are Configurable, Re-usable Models of Requirements and Designs

Results and Conclusions

In conclusion:

1. Prose requirements statements have an important role to play as a part of future model-based requirements data structures, as generalizations of transfer functions. This unifies a traditional requirements-writing skill with emerging model-based engineering techniques.

2. Requirements statements can be written in every-day natural language to explicitly refer only the system inputs, outputs, relationships between them, and parametric attributes of those relationships.
3. This improves the ability to write, understand, inspect, and use prose requirements statements, and improves the usual discipline of writing requirements statements, while maintaining traditional principles of requirements.
4. This approach also unifies the incorporation of requirements prose with other forms of input-output relationships, including equations, tables, graphs, and other relations.
5. This approach also improves the ability to create requirements patterns—libraries of configurable, re-usable requirements, improving the performance of the engineering process across larger product line and COTS enterprises.
6. Automated modelling and requirements tools can increase in their capabilities using this paradigm. We have applied this approach using the systems engineering and modelling tools of a number of tools suppliers.
7. Less experienced engineers can apply these concepts to improve their requirements writing and modelling. We have successfully taught this approach to undergraduate and graduate engineering students, as well as practicing engineers in commercial and mil-aero organizations.

References

1. Alexander, Christopher; Ishikawa, Sara; Fiksdahl-King, Ingrid; Angel, Shlomo, *A Pattern Language: Towns, Buildings, Construction*, Oxford U. Press, New York, 1977.
2. AP233 (ISO 10303) Web Site, `http://step.jpl.nasa.gov/AP233/,2004.`
3. Ashenden, Peter J., *The Designer's Guide to VHDL*, Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1996.
4. Booch, Grady; Rumbaugh, James; Jacobson, Ivar, *The Unified Modelling Language User Guide*, Addison Wesley, Reading, MA, 1999.
5. Buede, Dennis M., *The Engineering Design of Systems: Models and Methods*. John Wiley & sons, Inc., New York, 2000.
6. Carnap, Rudolf, *Introduction to Symbolic Logic and Its Applications*, Dover Publications, New York, 1958.
7. Chomsky, Noam; Piaget, Jean; et al, *Language and Learning: The Debate Between Jean Piaget and Noam Chomsky*. Edited by Massimo Piattelli-Palmarini. Harvard University Press, Cambridge, MA, 1980.
8. Chung, L.; Nixon, B. A.; Yu, E.; Mylopoulos, J., *Non-Functional Requirements in Software Engineering*, Springer Publishing, 1999.
9. Churchill, Ruel V., *Operational Mathematics*, McGraw-Hill Book Company, New York, 1958.

10. Clausing, D., and Hauser, R., “The House of Quality", *Harvard Business Review*, May-June, 1988.

11. Daniels, Jesse, and Bahill, Terry, “The Hybrid Process That Combines Traditional Requirements and Use Cases”, *Systems Engineering*, vol. 7, no. 4, 2004, pp 303-319.

12. Escher, M. C., *M. C. Escher: The Graphic Work*, Benedikt Taschen Verlag Publishers, 1992.

13. Gamma, E., Helm, R., Johnson, R, Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

14. Glasgow, Janice; Narayanan, N. Hari; Chandrasekaran, B., eds., *Diagrammatic Reasoning: Cognitive and Computational Perspectives*, MIT Press, Cambridge, MA, 1995.

15. Hamilton, William Rowan, "On A General Method In Dynamics" *Philosophical Transactions of the Royal Society (*Part II, 1834), pp 247-308.

16. Hayakawa, S. I., *Language in Thought and Action*. Fifth Edition. Harcourt Brace Jovanovich, Publishers, New York, 1990.

17. INCOSE MBSE, INCOSE Model Driven System Design Working Group web site, `http://www.incose.org/practice/techactivities/modelingtools/mdsdwg.aspx`, 2004.

18. Karayanakis, Nicholas M., *Computer-Assisted Simulation of Dynamic Systems with Block Diagram Languages*, CRC Press, Inc., Boca Raton, FL, 1993.

19. Landau, L. D.; Lifshitz, E. M., *Mechanics*, Third Edition. From *Course of Theoretical Physics*, *Volume 1*. Elsevier Science Ltd., Oxford, 1976.

20. Marcus, Gary F., *The Algebraic Mind: Integrating Connectionism and Cognitive Science*. MIT Press, Cambridge, MA, 2001.

21. Mellor, Stephen; Balcer, Marc J., *Executable UML: A Foundation for Model-Driven Architecture*, Addison-Wesley Publishers, Boston, MA, 2002.

22. Newton, Isaac, *Mathematical Principles of Natural Philosophy*, 1668, trans. and ed. by Andrew Motte, 1729; rev. by Florian Cajori, U. of California Press, Berkley, CA, 1934.

23. Oliver, David, *Engineering Complex Systems*, McGraw-Hill, New York.

24. Pinker, Steven, *The Language Instinct: How the Mind Creates Language*. William Morrow & Co., New York, 1994.

25. Schindel, W., and Smith, V., “Results of Applying a Families-of-Systems Approach to Systems Engineering of Product Line Families”, SAE International, Technical Report 2002-01-3086, November, 2002.

26. Schindel, William D., “Does Our SE House Have a Foundation?”, INCOSE Crossroads of America Chapter technical program presentation, Peoria, IL, May 22, 2002.

27. Schindel, William D., “System Engineering: An Overview of Complexity’s Impact”, SAE International, Technical Paper 962177, October, 1996.

28. Sussman, Gerald Jay; Wisdom, Jack, *Structure and Interpretation of Classical Mechanics*. MIT Press, Cambridge, MA, 2001.

29. SysML Partners Web Site, `http://www.sysml.org/ , 2004.`

30. Whorf, Benjamin Lee, *Language, Thought, and Reality: Selected Writings of Benjamin Lee Whorf*. John B. Carroll, editor. MIT Press, Cambridge, MA, 1956.

31. Zadeh, L. A., "Fuzzy Sets" *Information and Control,* vol. 8, 1965, pp 338-353.

Biography

William D. Schindel is president of ICTT, Inc., a systems engineering company, and the developer of the Systematica™ Methodology for model and pattern-based systems engineering. His 35-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, 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.
