Foundational Systems Engineering (SE) Patterns for a SE Pattern Language

Joseph J. Simpson
System Concepts
6400 32nd Avenue N.W., #9
Seattle, WA 98107
jjs-sbw@eskimo.com

Mary J. Simpson
System Concepts
6400 32nd Avenue N.W., #9
Seattle, WA 98107
mjs-sbw@eskimo.com

Abstract. Patterns are classically used to effectively capture large, complex bodies of information and knowledge. This paper outlines the current state of systems engineering (SE) pattern literature as well as proposing a foundational set of SE patterns for use in SE pattern languages. A pattern is defined as a solution of a specific problem placed in a specific context. A pattern language is a collection of interrelated patterns, with specific relationships binding individual patterns together. A collection of SE pattern relationship types are also introduced in this paper. Language is a fundamental aspect of any technical or scientific activity. The technical language used in technical activities impacts the concepts that can be addressed, the effort to express these concepts, and the precision with which these concepts can be expressed. Therefore it is important to consider the language requirements for conceptual expression and precision when designing a new language. In the case of an SE pattern language, the elements to be considered are the individual patterns and the relationships between the patterns.

System Engineering (SE) Patterns Introduction

SE patterns have been discussed in INCOSE literature by at least two authors in three papers. "A Systems Engineering Pattern Language," a paper by Robert H. Barter that appeared in the 1998 INCOSE Symposium Proceedings, first proposes the creation and use of a systems engineering pattern language. His paper suggests that SE patterns and pattern languages should be based on the capture of SE best-practices, and formal SE processes as well as the Systems Engineering Book of Knowledge (SEBOK). The paper proposes using patterns and pattern languages to capture and manage the SEBOK information. During his presentation on SE patterns, Barter suggested that SE patterns should also be based on Willoughby Templates. In 2003, Cecilia Haskins again proposes the use of SE patterns to capture the information in the SEBOK. There is a difference between the definition and development of pattern languages between these two first INCOSE pattern papers. Barter suggests the use of 'Concept Map'-like relationships between the individual patterns in a SE pattern language, while Haskins refers to a pattern language as a collection of patterns, and does not address the specific relationship types between patterns in any detail. In a 2005 paper, "Application of Patterns and Pattern Languages to Systems Engineering," Haskins addresses the relationships component of pattern languages in a different manner by addressing links between patterns at different levels and appealing to the ideas of networks and hierarchy to describe these links (Haskins 2005).

Pattern Definition. A pattern is defined as a solution of a specific problem placed in a specific context. Both the (Barter 1998) and (Haskins 2003) papers describe the form of patterns as

containing: name, problem statement, forces, context, and a solution. Barter included 'related patterns' as part of a pattern form, whereas Haskins included 'resulting context' as a part of a pattern form

> **"Resulting Context.** This section contains a description of the state of the world after the pattern has been applied. This part of the pattern should describe the impact of applying the pattern, both positive and negative. Potential users of the pattern can study this section to weigh the potential costs and benefits. In a pattern language, the resulting context of one pattern often becomes the context for successive patterns that address new issues (tensions)" Haskins (2003).

It appears from the Haskins definition of 'resulting context,' that a "part of" hierarchy is implied as a relationship that applies to most patterns. Part of the intent of this paper is to focus on clarification of the relationships between patterns. As a result, the form expressed by Barter shall be adapted and proposed for standard use (see Figure 1.). *Evocative name* is the title that briefly highlights and/or brings to mind the particular pattern being addressed. *Problem* – or problem statement – expresses the primary difficulties and uncertainties in the situation. *Forces* articulate the tensions or key drivers that influence the application of a given pattern. *Context* reflects the meta-system abstraction frame, and is set by the environment system and the production system (Simpson and Simpson 2005). *Related patterns* are those patterns that may be connected in some way to the current pattern. *Solution* expresses how the problem is resolved in the specific context.

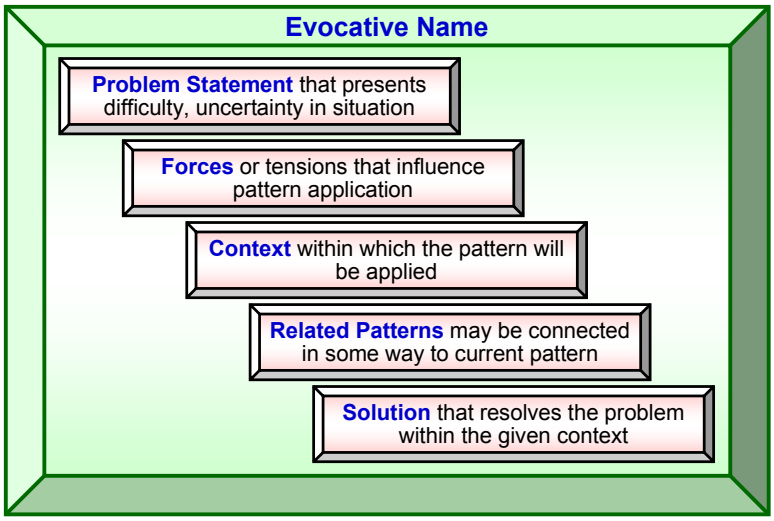

Figure 1. Pattern Form

Pattern Language Definition. A pattern language is a collection of interrelated patterns, with specific relationships binding individual patterns together. There is a substantial range or continuum of possible pattern languages. Since the patterns remain the same, the formality of a pattern language depends on the types of relationships between its patterns. With this in mind, classes of system pattern relationships, ranging from informal to formal types of connectivity, will be examined.

Global Systems Engineering Patterns

Three high-level, global patterns have been identified that can be used as a means of organizing systems patterns. (1) Anything can be described as a system. (2) The problem system is always separate from the solution system. (3) Three systems, at a minimum, are always involved in any system activity: the environmental system, the product system, and the process (that produces the product) system. It is recognized that extensive descriptions exist that augment these particular patterns. The intent is to create an initial 'ontology' of high-level pattern descriptions that provide a framework within which pattern relationships can be discussed, and proposed as a part of a systems pattern language.

Anything Can Be Described as a System. A description exists – iteratively throughout literature – of a pattern that can be called *General Systems*. (Mar and Morais 1997) allude to the general systems theory that "presents a model that can be applied to describe anything in terms of its boundary, its inputs and outputs and the transforms used to convert inputs to outputs." That pattern has been documented briefly in the form shown in Figure 2 – General Systems Pattern

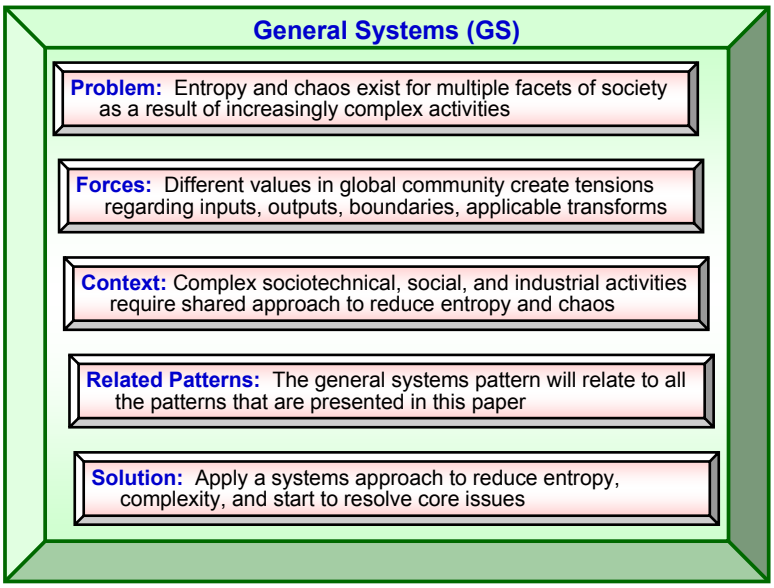

Figure 2. General Systems Pattern

Problem System Is Separate from the Solution System. Multiple examples of the *Problem vs Solution* pattern exist. Figure 3 briefly documents the pattern that separates the problem system from the solution system. (Goode and Machol 1957) separate the problem system from the solution system by referring to the Exterior and the Interior steps in system design. 'Exterior' contains and describes the statement of the problem. 'Interior' addresses the suggested solution system. (Mar and Morais 1997) are careful to describe – as a part of establishing the boundaries of the system – that "This definition is critical to formulating the problem to be solved, and to develop a shared vision of a common problem." (Warfield 2002), in his discussion of the "Work Program of Complexity," clearly separates the problem from the solution by use of the terms Discovery and Resolution. Discovery reflects the need to describe and to diagnose the problem situation. Resolution incorporates the actual design and implementation of the solution.

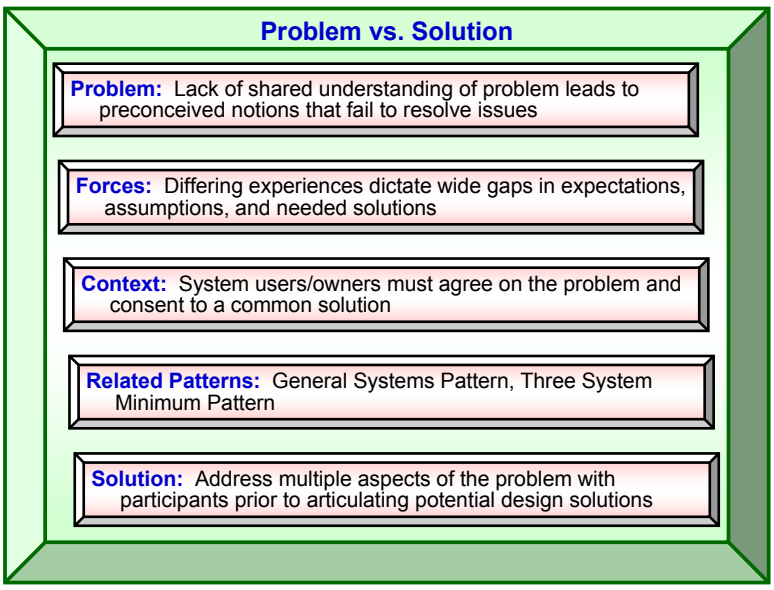

Figure 3. Problem vs. Solution Pattern

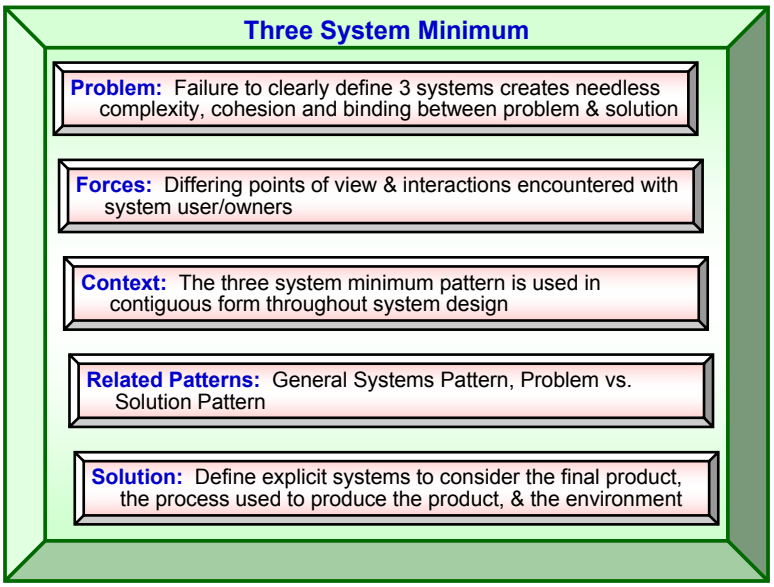

Figure 4. Three System Minimum Pattern

Three System Minimum Pattern: Systems for Process Development, Product Development, and Environment Description. (Mar and Morais 1997) described the need for a product system a process (or Program) system that is used to produce the product, an operating environment system that includes normal, abnormal operating as well as start up and shut down environments, a program environment system that includes the regulators, investors, and competition and the "Rest of the World" System. These systems were simplified over time to three basic systems, and were expressed by (Simpson and Simpson 2003) as the product system, the process/ organizational system, and the environment system. Figure 4 – *Three System Minimum* Pattern – briefly documents the pattern that reflects these systems.

Examples of Basic SE Patterns

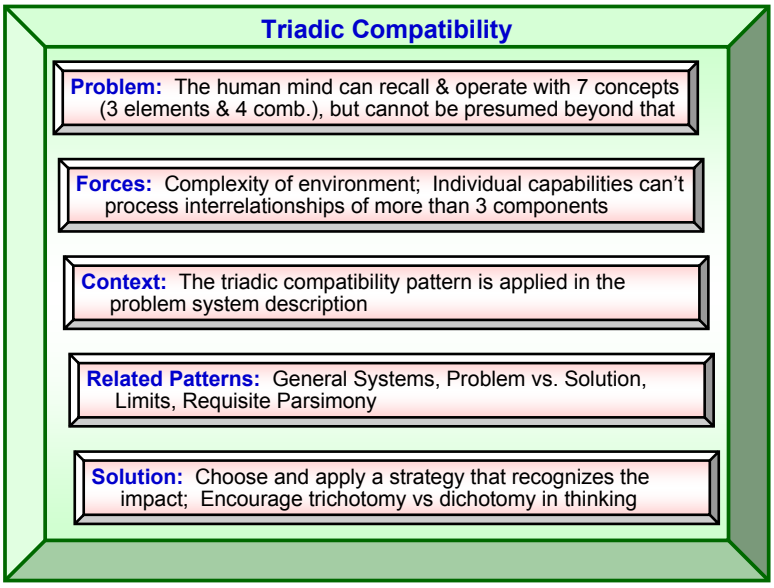

Figure 5. Triadic Compatibility

Triadic Compatibility Pattern. The *Triadic Compatibility Pattern* is based on the brief for the Law of Triadic Compatibility proposed by (Warfield 2002). This pattern addresses the idea that the human mind is incapable of recalling into its short-term memory more than about seven items. (A set comprised of three elements and the four interacting combinations of them will consist of seven members.) In addressing complex systems, it is essential that this pattern reflecting human limitations and capabilities be considered. Figure 5 briefly describes the Triadic Compatibility Pattern.

Requisite Parsimony Pattern. This pattern is based on the dynamics of interpreting and learning implied by the law of triadic compatibility applied to the designer in the solution system. (Warfield 2002) postulated that if the designer is not operating at the 'Limit of reasoning capability,' (i.e., is either underburdened or overburdened), that the "design Target ... will embody bad outcomes that are beyond the control of the designer." He further postulated that if the designer is overburdened, no reliance can be placed on the designer's decisions. Figure 6 briefly describes the *Requisite Parsimony* Pattern.

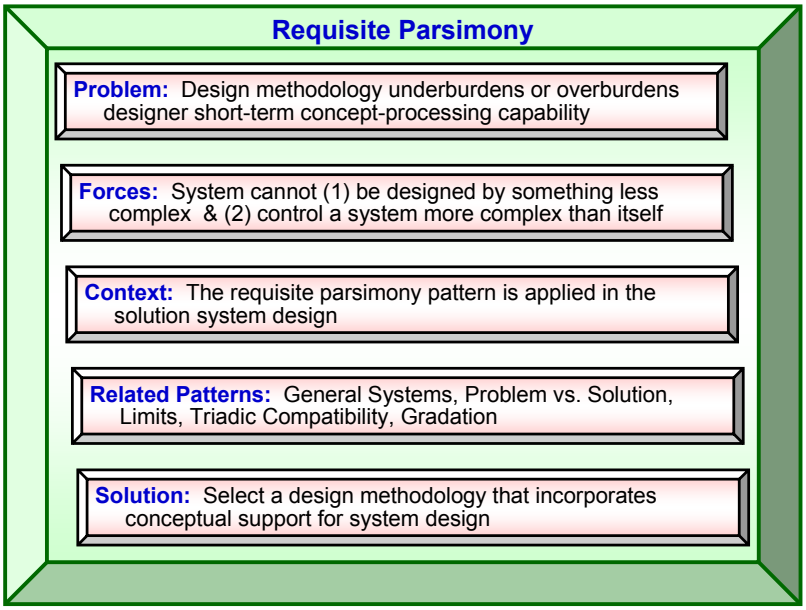

Figure 6. Requisite Parsimony

Limits Pattern. The *Limits* Pattern is based on (Warfield 2002) Brief 11, The Law of Limits. This law is based on the proposition that any activity will have a corresponding set of limits on that activity that will determine its feasible extent. Four corollaries are named as a part of this law: Active Limits, Movable Limits, Discretionary Action, and Shifting Limits. This law, and its four corollaries, suggest that awareness and an explicit description of a systems limits can substantively contribute to the effort of describing and analyzing the problem, and ameliorate potential impacts to the system. Figure 7 describes the Limits Pattern.

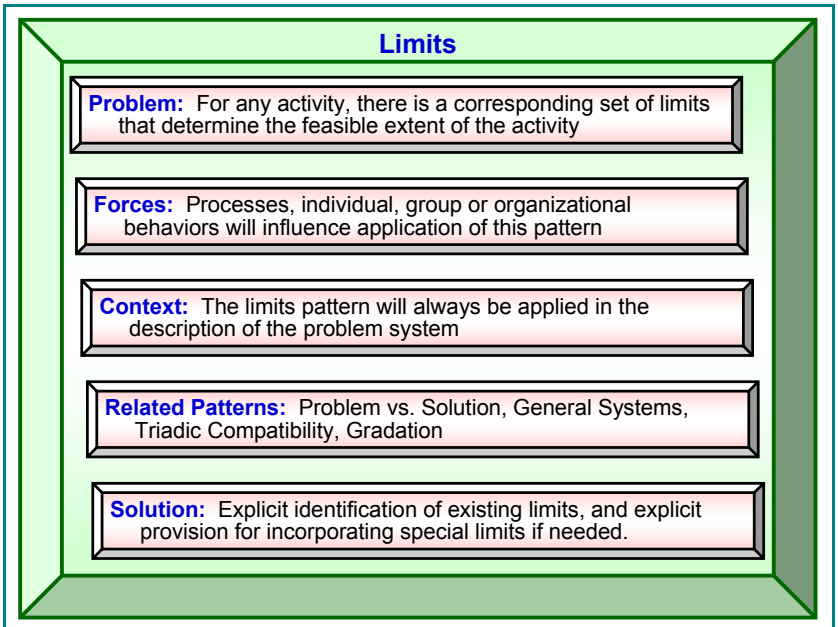

Figure 7. Limits Pattern

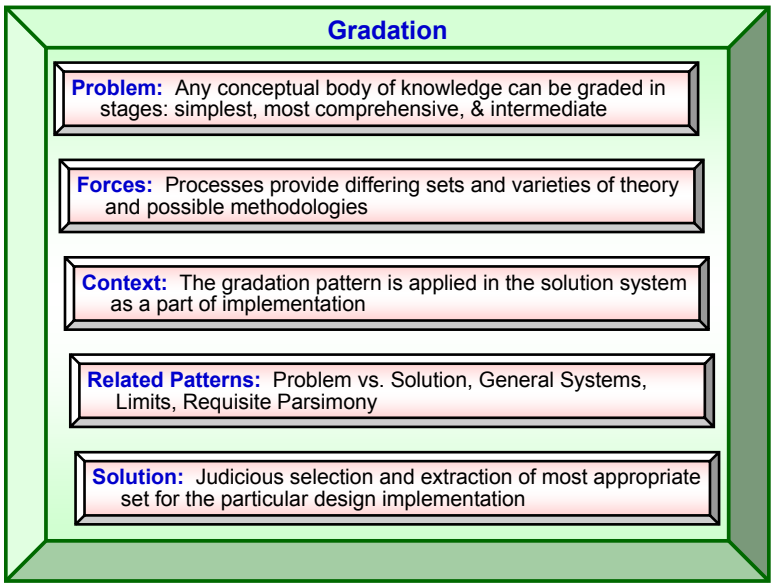

Figure 8. Gradation Pattern

Gradation Pattern. The *Gradation* pattern, based on (Warfield 2002) Brief 8, The Law of Gradation, is briefly described in Figure 8. Gradation speaks to the fact that it is not credible to force any system to deal with the entire conceptual body of knowledge. It recognizes the fact that system solutions may range from "very small, limited-scope Targets to very large, broad-scope Targets." This brief goes on to state:

"It is **not** reasonable to take as a criterion for Generic Design Science that all of its Theory and all of its Methodology should be demonstrably required for all design activity. On the contrary, such a Science would be too brittle for use. The Law of Gradation overtly recognizes that Design Situations and Design Targets are themselves graded according to a variety of descriptions, not all of which can be foreseen. Accordingly the Science of Generic Design should be applied judiciously, extracting from it one of its stages that is most appropriate for the particular Design Situations and Design Target."

Corollaries associated with the Gradation Law include that of Congruence, Diminishing Returns, and Restricted Virtual Worlds.

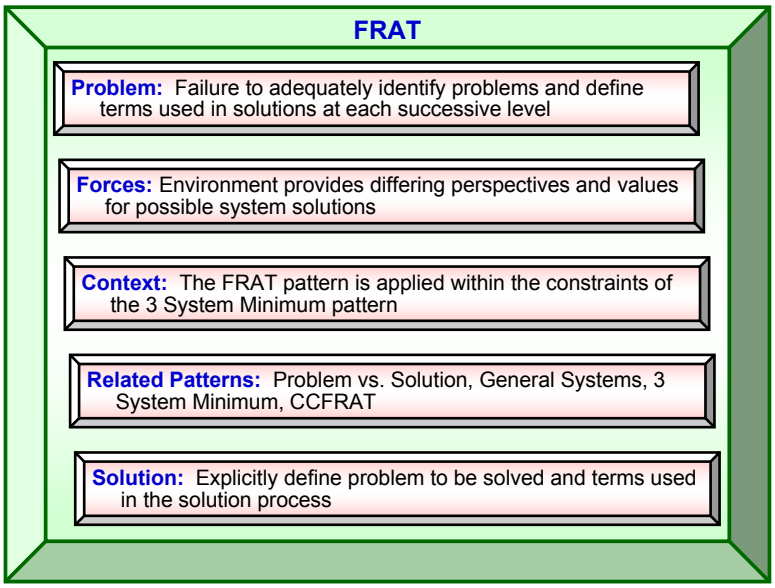

Figure 9. FRAT

Functions, Requirements, Architectures and Tests (FRAT). The FRAT pattern, described in Figure 9, is based on the general problem solving process, and was developed by Brian W. Mar. FRAT stands for **F**unctions, **R**equirements, **A**nswers, and **T**ests. 'Functions' refers to what the system needs to do. 'Requirements' refers to how well the system must perform these functions. 'Answers' refers to the set of alternatives and the recommended architecture that represents how the systems will perform. 'Tests' refers to the way in which the architecture is verified to meet the functions. Functions and Requirements address the problem space and together determine WHAT is needed to solve the problem. Answers and Tests address the solution space, and together determine HOW the problem is solved (Mar 1997).

Context, Concept, Functions, Requirements, Architectures and Tests (CCFRAT). By emphasizing the system boundary, the CCFRAT pattern, briefly described in Figure 10, focuses on external system forces in the context portion, and focuses on the internal system forces in the concept portion (Simpson and Simpson, 2005). **C**ontext is the outward looking boundary of the system and is set by the value constraints imposed by the environment system and the production system. **C**oncept is inward looking from the system boundary and defines the conceptual relationships within the system. Within the established concept, FRAT may be used to define the problem and solution systems. As in FRAT, 'Functions' defines what the system does, 'Requirements' defines how well the functions are done, 'Answers' defines how the function is performed, and 'Tests' defines how you know that the answer does the function as well as the requirement states.

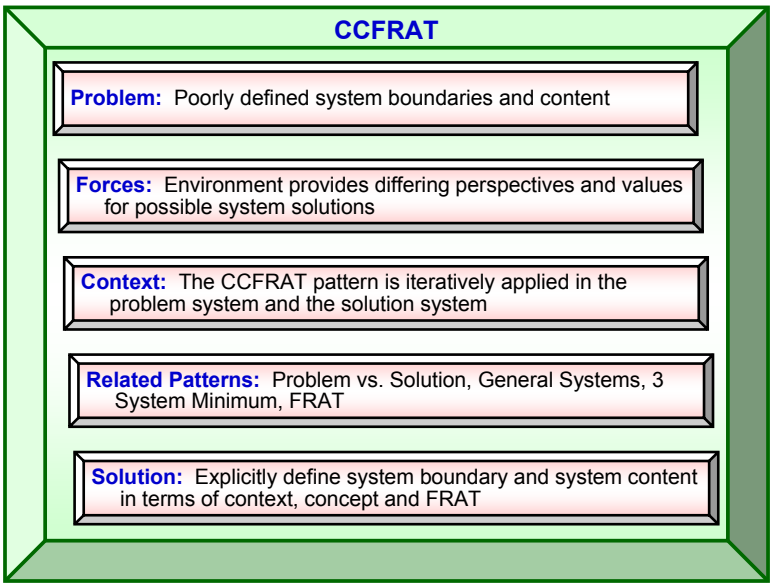

Figure 10. CCFRAT

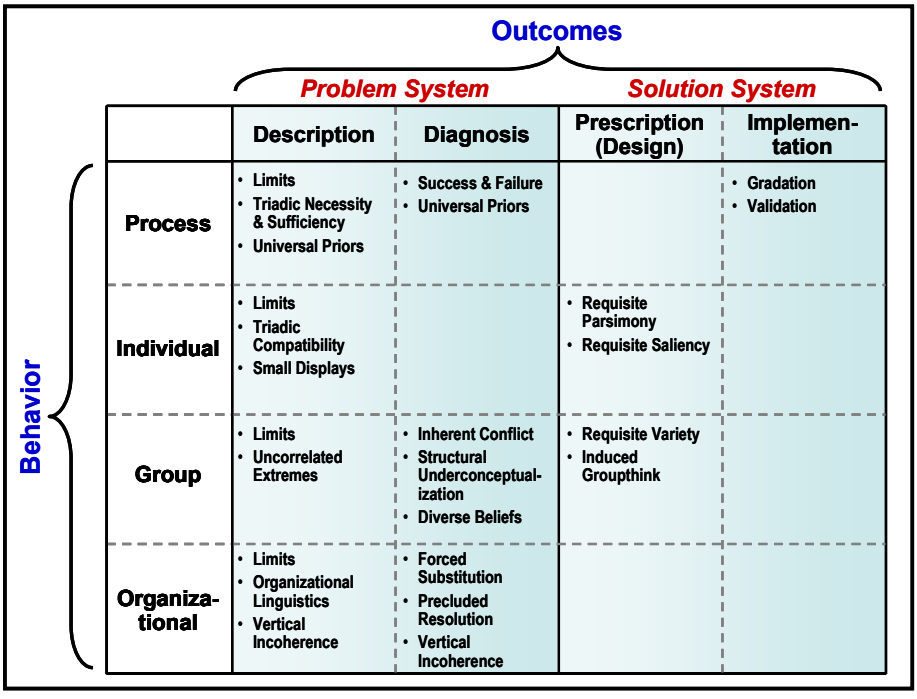

Figure 11. Warfield - Behavior Outcomes Matrix

Components of a SE Pattern Language

Patterns and Relationships. Pattern languages are made up of two components, patterns and relationships between patterns. The relationships between patterns can take a number of forms and must allow a logical, natural combination of patterns. Figure 11 shows some of the relationships between Warfield's patterns.

Examples of SE Pattern Language Relationships

There is a substantial range or continuum of possible pattern languages. The formality of a pattern language depends on the relationship types – because the patterns remain the same. In discussing language as related to systems, Warfield (1990) noted that "it must be possible to make associations and assignments between intuitively-generated relationships expressed through natural language and formal statements of these expressed in terms of the object language." Figure 12, adapted from Warfield (1990), is a notional depiction that moves from an informal type of connectivity to a formal one. Simpson and Simpson (2005) indicated that "the set of system meta-levels and meta-level transforms must be formalized in a structured fashion to support the development of a systems engineering language."

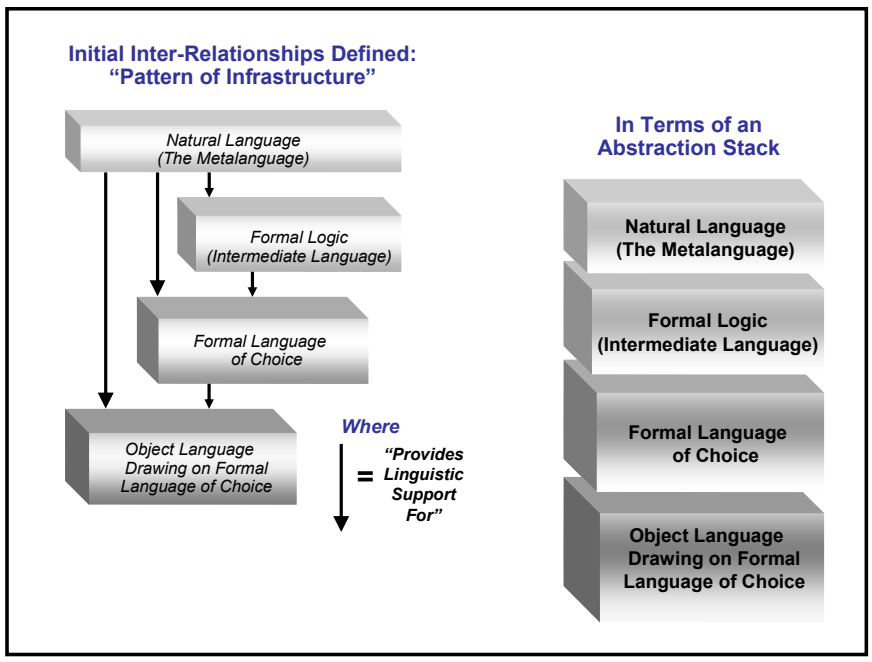

Figure 12. Systems Language Design Capability

Concept Map Style. Related patterns were included in the system patterns proposed by Barter (1998). These related patterns were connected to form a pattern language. Barter suggested the use of concept map notation for the representation of relationships between patterns. This notation provides for naming the type of relationship between patterns and provides the means to graphically represent the pattern language as a pattern map.

Sequential Concept Map Style. If the concept map notation is used to develop pattern languages and pattern maps then there are many styles of concept map relationship notation that can be used for this task. McCartor and Simpson (1999) outlined some of the basic relationship types including, top down, center out and sequential relationship tags. These relationships provide a wide variety of possible combinations between system engineering patterns. The primary advantage of the sequential relationship style is the ability to design a system engineering pattern language that is a process that includes concurrent and sequential relationships. All of the concept map notations are designed reading and interpretation by human beings.

Hierarchical Style. There are a number of hierarchical relationship styles that can be used effectively in a systems engineering pattern language. One of the most important is a "recursive hierarchical" relationship. An example of this hierarchical relationship, shown in Figure 13, is the recursive application of the FRAT pattern to a system design and development problem. The FRAT pattern is applied at every level of system abstraction, design and/or decomposition. Another important recursive pattern relationship application is the CCFRAT pattern which is used recursively to develop the system boundaries at each level of system interaction, abstraction, and/or design.

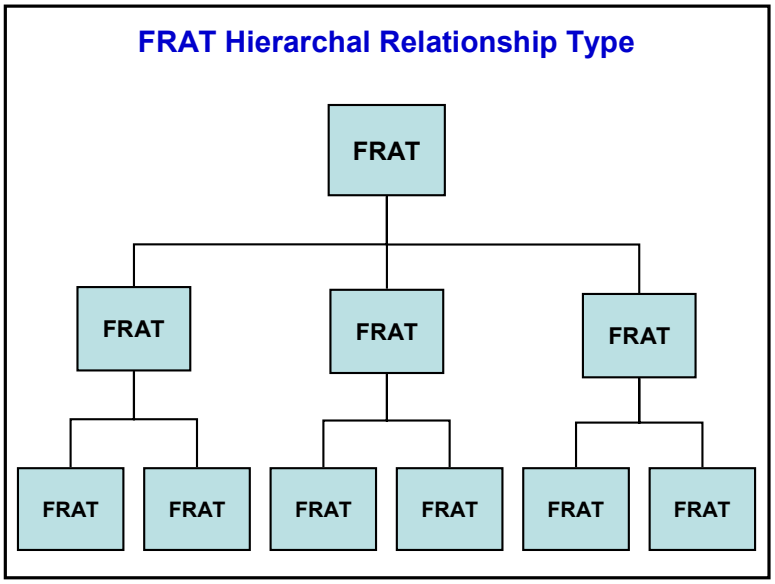

Figure 13. FRAT – Recursive Hierarchy

Two of the primary components of FRAT are functions and answers (or architecture). Both of these components have classically been handled using some type of hierarchical model or pattern. Functional analysis creates a function hierarchical pattern that has equivalent functional behavior at each level in the functional abstraction hierarchy. A hierarchical pattern created using this functional analysis approach has several interesting and useful characteristics. Two of these characteristics are depicted in Figure 14: (1) clear abstraction level identification and (2) level completeness operators.

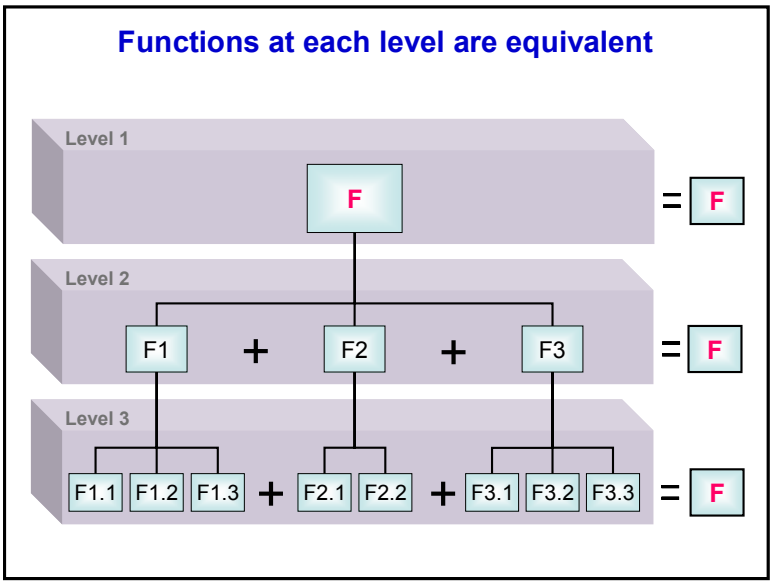

Figure 14. Function Hierarchy

Clear abstraction level identification is used during the system design process to communicate among design teams and other interested stakeholders. The abstraction level completeness operator is used to evaluate the contents of the current abstraction level and determine if the functional analysis is complete. The completeness operator is a valuable conceptual tool for functional system design evaluation and analysis.

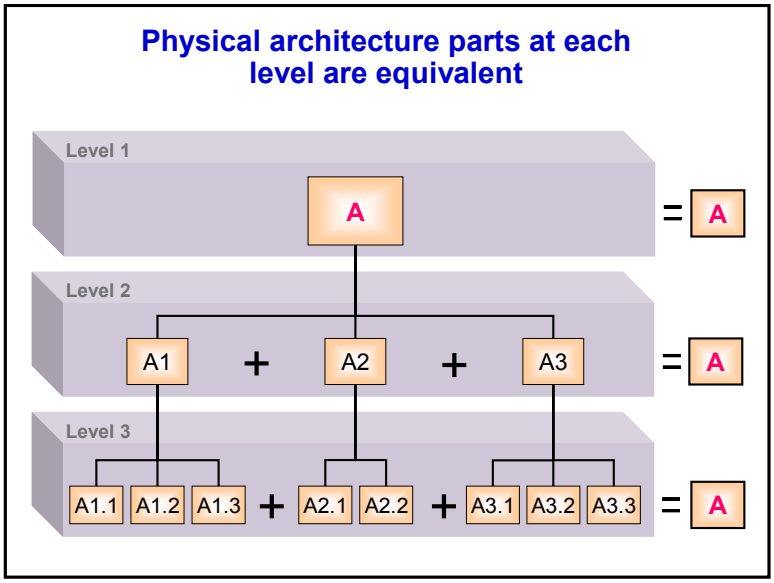

Figure 15. Physical Architecture Hierarchy

Physical architecture is one of the most common types of system representation, with the physical decomposition (or part of) hierarchy being one common type of engineered system representation (see Figure 15). Like the function hierarchy pattern, the physical hierarchy pattern has clear levels of abstraction and completeness operators. However, the fundamental concept used for the decomposition is different in each case. When these two decomposition techniques are applied to the same system the resulting hierarchies almost never have the same levels of abstraction. The SE hierarchical pattern relationship can be applied in various ways to fulfill a range of design goals and objectives. However, these hierarchical relationships may have different meanings as they are applied in the practice of systems engineering (see Figure 16). Mapping between and among each specific hierarchy is necessary to convey the integrated aspects of the systems objectives, goals and behavior.

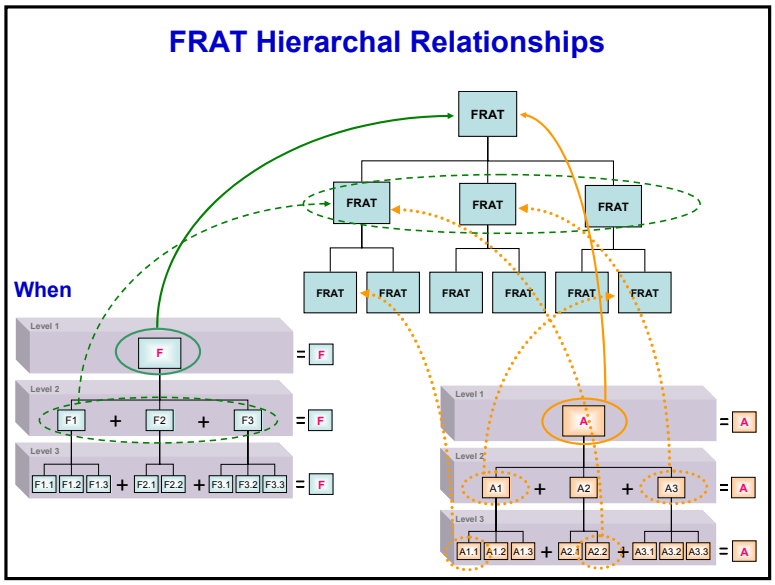

Figure 16. Hierarchical Mapping Relationships

The FRAT pattern has been used to emphasize the importance of relationships in an SE pattern language. In this case a self-similar, recursive hierarchical pattern relationship was used to create a description of a system. Then, two other types of hierarchical relationships were used to create a view of the system under consideration.

It is clear that the FRAT SE pattern can be connected using a hierarchical relation to create a FRAT SE pattern language. Further discussion and evaluation is needed to determine if the function hierarchy and physical architecture hierarchy should be considered as stand alone SE patterns, or as SE pattern language relationships. It is clear that hierarchical mappings are used in some SE patterns. Therefore a mechanism must be developed to distinguish the interface between the SE pattern application of hierarchy and the

hierarchical relationship used to create an SE pattern language.

Sequential Framework Style. The sequential pattern relationship style is used to develop a pattern language that has concurrent multiple levels of abstraction and a guiding sequential system framework. This complex SE pattern relationship style provides specific areas for discovery patterns, knowledge patterns, design patterns and system patterns encapsulated in a set of sequential logic frames. The system abstraction framework shown in Figure 17 was developed and described by Simpson and Simpson (2005). This type of relationship supports the development of a pattern language to express both concurrent and sequential types of relationships between the selected SE patterns.

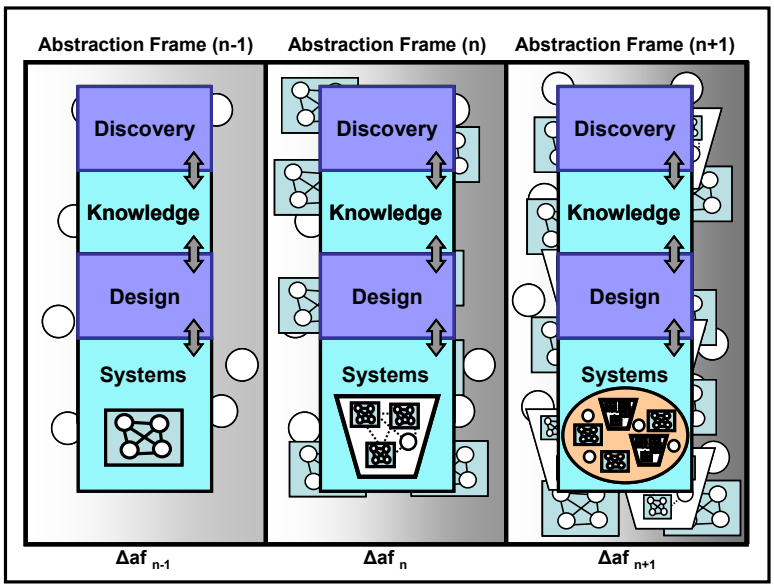

Figure 17. Sequential Frameworks

Abstraction Stack Style. The abstraction stack pattern relationship style provides the ability for the pattern language to express a well defined set of abstractions between using one organizing concept among patterns. This relationship type can be combined with the hierarchical relationship style to express a hierarchical set of abstraction stack types of relationships in a systems engineering pattern language. Figure 18 provides an example of the system abstraction stack relationship type.

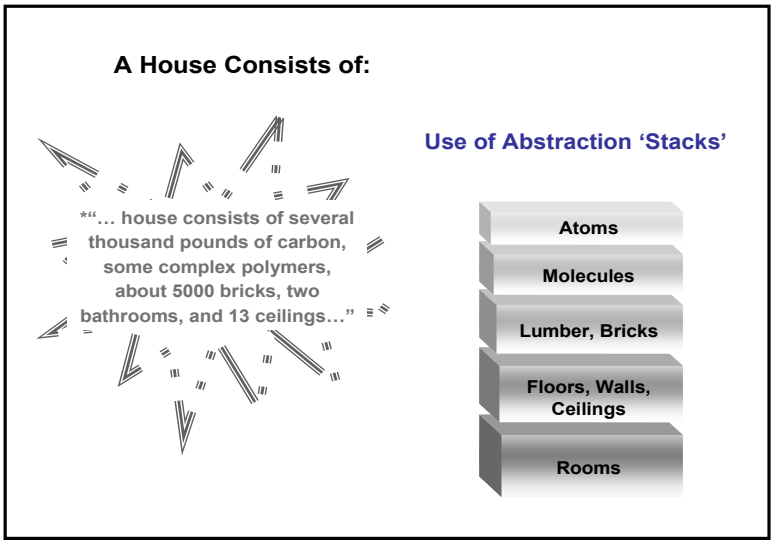

Figure 18. Abstraction Stacks

Summary and Conclusions

Well established and understood systems engineering patterns provide the foundation for the communication of complex systems concepts. A core set of systems engineering patterns has been presented in this paper to stimulate the development and discussion of systems engineering patterns. The related patterns component is a necessary part of the pattern template and provides the basis for the construction of pattern languages using related patterns and

specific types of relationships. Types of pattern relationships are introduced as a mechanism to categorize and standardize basic relationship types. The patterns presented here are high level patterns used to set the global systems engineering pattern language context. The core pattern set and standard relationship connections can be used to communicate very complex sets of system information. SE pattern relationships range from informal to formal, with the more formal types of relationships providing the foundation for an executable SE language.

More research is needed to further develop the core set of systems engineering patterns and pattern relationship types. This research should also cover groups of related patterns or pattern languages.

REFERENCES

Barter, Robert H., "A Systems Engineering Pattern Language." *Proceedings of the 8th Annual International Symposium of the International Council on Systems Engineering* (Vancouver, BC, July, 1998).

Dahlberg, Sten, "Functional Analysis Tutorial." *Tutorial Proceedings from the INCOSE Seattle Metropolitan Chapter Tutorial Series* (Kent, WA, October 3, 2005).

Goode H., Machol R., *Systems Engineering, An Introduction to the Design of Large-scale Systems*. McGraw-Hill Book Company, New York, 1957.

Haskins, Cecilia., "Using Patterns to Share Best Results – A Proposal to codify the SEBOK." *Proceedings of the 13th Annual International Symposium of the International Council on Systems Engineering* (Washington, DC, July, 2003).

Haskins, Cecilia., "Application of Patterns and Pattern Languages to Systems Engineering." *Proceedings of the 15th Annual International Symposium of the International Council on Systems Engineering* (Rochester, NY, July, 2005).

Hitchins, Derek K., *Advance Systems Thinking, Engineering and Management.* Artech House, Inc., Boston, MA, 2003.

Mar B. W., "Back to Basics Again, A Scientific Definition of Systems Engineering." Proceedings of the Seventh Annual International Symposium of the International Council on Systems Engineering (INCOSE-97). Los Angeles, CA, 229-236.

Mar, B.W., and Morais, B.G., *The Engineering of Complex Systems*. University of Washington Reference Text # R819-UW/SAI, Seattle, WA, 1997.

McCartor, M.M., and Simpson, J.J., "Concept Mapping as a Communications Tool in Systems Engineering." *Proceedings of the 9th Annual International Symposium of the International Council on Systems Engineering* (Brighton, England, June, 1999).

Simpson, J.J. and Simpson, M.J., "Systems and Objects." *Proceedings of the 13th Annual International Symposium of the International Council on Systems Engineering* (Washington, DC, July, 2003).

Simpson, J.J. and Simpson, M.J., "System Integration Frameworks." *Proceedings of the 15th Annual International Symposium of the International Council on Systems Engineering* (Rochester, NY, July, 2005).

Warfield, John N., *A Science of Generic Design.* Iowa State University Press, Ames IO, 1990.

Warfield, John N., *Understanding Complexity: Thought and Behavior.* Ajar Publishing Company, Palm Harbor F, 2002.

BIOGRAPHY

Joseph J. Simpson's interests are centered in the area of complex systems including system description, design, control and management. Joseph has professional experience in several domain areas including environmental restoration, commercial aerospace and information systems In the aerospace domain, Joseph has participated in a number of system development activities including; satellite based IP network design and deployment, real-time synchronous computing network test and evaluation, as well as future combat systems communications network design. Joseph Simpson has a BSCE and MSCE from the University of Washington, an MSSE from the University of Missouri-Rolla, is a senior member of IEEE, and a member of INCOSE, and ACM. Currently Joseph is enrolled in a system engineering doctorate program at the University of Missouri-Rolla.

Mary J. Simpson's experience and interests focus on cognitive support and decision-making systems, integration and complexity reduction, threat and vulnerability analyses, systems sciences and systems engineering. Mary applies systems solutions to complex problems encountered in organizations, processes, and systems interactions ranging from strategic to tactical levels in fields including aerospace, security, information systems, energy, tank waste, public involvement, emergency planning, biological and chemical systems, and defense systems. Mary is currently applying her leadership and integration capabilities to multiple opportunities in science and technology, homeland security, quality risk assessments, defense systems and systems sciences.