02TB-87

Results of Applying a Families-of-Systems Approach to Systems Engineering of Product Line Families

William D. Schindel
ICTT, Inc. and System Sciences, LLC

Vernon R. Smith
Caterpillar Inc.

ABSTRACT

Most of the history of systems engineering has been focused on processes for engineering a *single* complex system. However, most large enterprises design, manufacture, operate, sell, or support not one product but *multiple* product lines of related but varying systems. They seek to optimize time to market, costs of development and production, leverage of intellectual assets, best use of talented human resources, overall competitiveness, overall profitability and productivity. Optimizing globally across multiple product lines does not follow from treating each system family member as an independently engineered system or product. Traditional systems engineering principles can be generalized to apply to families. This article includes a multi-year case study of the actual use of a generic model-based systems engineering methodology for families, Systematica™, across the embedded electronic systems products of one of the world's largest manufacturers of heavy equipment.

INTRODUCTION

Systems Engineering has been formally recognized long enough to have accumulated a wealth of literature [1, 4, 53] and definitions. A typical definition is "Systems Engineering is an interdisciplinary approach and means to enable the realization of successful systems. It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements, then proceeding with design synthesis and system validation while considering the complete problem: Operations, Performance, Test, Manufacturing, Cost & Schedule, Training & Support, Disposal. Systems Engineering integrates all the disciplines and specialty groups into a team effort forming a structured development process that proceeds from concept to production to operation. Systems Engineering considers both the business and the technical needs of all customers with the goal of providing a quality product that meets the user needs." [53]

Another way to think about the purpose of systems engineering [2, 3, 52] is that it attacks the problems we encounter as general man-made systems become increasingly complex. It becomes harder to do the following with systems, as their complexity increases:

Describe,	*Understand,*	*Communicate about,*
Predict,	*Install/deploy,*	*Design and implement,*
Manage,	*Operate,*	*Manufacture,*
Monitor,	*Repair,*	*Diagnose,*
Configure,	*Maintain,*	*Control,*
Evolve,	*Account for,*	*Maintain security of,*
Integrate,	*Validate,*	*Verify.*

By "harder", we mean measured in terms of cost, effort, schedule, or risk.

This paper describes an approach to the problem of complexity that arises from product lines. We call these "families of systems" in the case of systems that are not viewed as products. This approach has additionally yielded benefits reported here related to other complexity drivers, such as embedding of intelligence in systems.

SYSTEMS ENGINEERING OF CONFIGURABLE PRODUCT LINES, OR FAMILIES OF SYSTEMS

SYSTEMS ENGINEERING'S SINGLE SYSTEM ORIGIN

Systems engineering is frequently described as originating in the mid-Twentieth Century, in defense

systems; others argue that it originated in the planning of much earlier man-made systems [4]. Although the field has an impressive history of accomplishments, problems encountered by seasoned systems engineering teams with very complex systems remind us that the discipline is still developing.

Understanding the nature of complexity growth encountered in systems sheds light on what is needed to manage it successfully. Two factors driving growth of complexity higher are:

1. Competition/Technology Syndrome: In competitive markets, advantage may go to products (or services) with higher perceived sophistication (behavior modes in the face of different environmental situations). The continuing arrival of new technologies is an "enabler" that combines with this market "hunger" for sophistication (features), frequently leading to growth in total system complexity. (Example: Embedded electronics and software added sophistication to engines, but raised the complexity of the total engine system, for development, service, etc.)
2. Product Line / Configurable Systems Syndrome: Economically-driven organizations are interested in leveraging assets to cover the widest possible range of applications. Whether we call these product lines or configurable families of systems, modern business has been pursuing economic leverage in this way since the days of Cyrus McCormick (International Harvester) and Alfred P. Sloan (GM). (Contemporary examples include product lines of engines and machines, along with configurable services such as telecommunications, transportation, and medicine.)

Systems engineering has been focused on *single* systems for much of its history. Ask a systems engineer how *multiple* systems are involved in systems engineering and the likely reply is that systems engineering can help integrate them to work together. But what if we want to solve recurring engineering problems globally across enterprise-wide families, or ensembles, of products having a measure of common content, but configurable for different markets, applications, or customers? This is not a problem of systems integration. It is *global optimization across families of similar systems*.

This is one of the leading economic challenges of our day, but has not been well addressed by an appropriate discipline aimed squarely at the families problem. An ironic indicator of the historical single-system focus of systems engineering is that EIA 632, a relatively contemporary cross-industry authoritative reference on systems engineering [42], is titled "Processes for Engineering ***A*** System" (author's emphasis).

THE PROBLEM OF PRODUCT LINES

More precisely, what do we mean by systems engineering of families of systems or product lines?

Product line pursuit arises by seeking improved *economic leverage*. The idea is to focus the enterprise's investment and expertise on one general type of product or system but to arrange for different specializations or configurations of that general system type to fit the specific requirements of different market segments, applications, or customers.

Example products lines or families of systems include:

- On road vehicle automotive product lines
- Aircraft product lines
- Switching system product lines
- Integrated circuit product lines
- Application software product lines
- Locomotive product lines
- Internal combustion engine product lines
- Personal computer product lines
- Families of telecommunication services
- Families of chemical processing plants
- Families of military operations
- Families of service business processes

Figure 1 illustrates the idea graphically, with diverse products across the base, organized into similar families (higher level), all part of a general type of system (higher still). This diagram could have several vertical levels, to illustrate the groupings of a particular enterprise. (A later section of this article describes a six layer example in a specific enterprise.) We want to focus our investment on the narrow (upper) part of the triangle of products, while earning revenue across the wide (lower) part of the triangle. This is the essence of economic leverage.

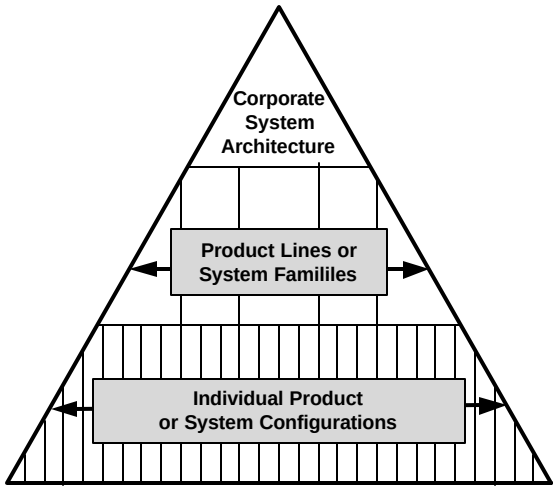

Figure 1: Product Lines/Families of Systems

However, our experience in talking with many companies shows us that this idealization is not what usually happens. Even though a diagram such as Figure 1 is frequently claimed to represent the product line structure of an enterprise, examining the actual engineering, manufacturing, distribution, and support practices of the enterprise tell a different tale. Practitioners frequently agree that the actual means of maintaining a degree of

similarity across products is movement of people and "cloning" of specifications, as shown in Figure 2.

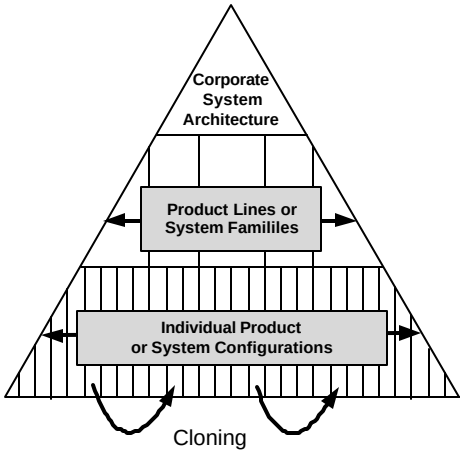

Figure 2: System "Cloning" Is A Typical Approach

This seems at first a pragmatic solution. However, it breaks down because while it creates many propagated specifications across the base of the triangle of Figure 2, it does not create controlling specifications at the higher levels of that same triangle. The upper levels of the triangle thus become intuitively understood but formally non-existent (to varying degrees). There is no real control structure such as the triangle implies, but instead just a large number of specifications across the bottom.

Over time, as dissimilar market pressures impact on the different product configurations at the base of the triangle in different market segments, the products are driven apart, just as they would be by physical pressures. As illustrated by Figure 3, the triangle begins to "fly apart" as the informal upper level architecture decays.

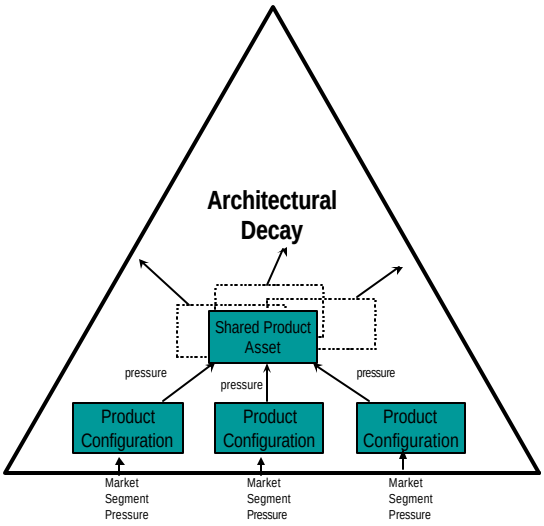

Figure 3: Forces of Family Architecture Decay

This decay in family architecture is enabled when there is a lack of "blueprints" for upper level family levels. This is because of lack of a uniform notation for both abstract and specific systems (especially for their behavior) and lack of a means of checking consistency between them. This encouraged our research into model-based means of enabling family management disciplines.

What we were seeking is the situation summarized by Figure 1--formal specifications of not only the products at the bottom of the triangle, but also the abstract architectures and common content at the upper layers. If this is successful, we would expect that over time instead of the decay of Figure 3 we would see the ongoing evolution of Figure 4. Upper level content would evolve slowly and lower level content more rapidly.

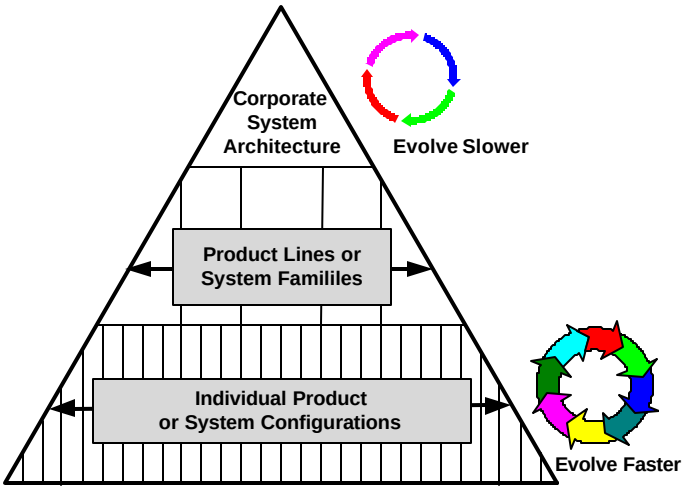

Figure 4: Different Rates of Evolution

FORMALIZING FOUNDATIONS OF SYSTEMS ENGINEERING USING MODELING METACLASSES

The families of systems problem described above was the motivation for research leading to a systems engineering formulation for families of systems. The results of this effort have had many other benefits, but the origin could be thought of as a need for "blueprints" in the diagram of Figure 1. We believed that the engineering "blueprints" (specifications) for systems across the bottom of this diagram had to be matched with similar blueprints for the upper level product lines and general system types. Without a blueprint for a family, how can we determine whether a proposed product conforms to the precepts of its product family?

The problem was to find a practical means of generating specifications that represented classes of systems, but that could be further specialized or configured. The resulting approach we developed, Systematica™, uses the ideas of model-based systems engineering [5-18]. We found that the use of models allowed a single

common “language” to be used by systems engineers to describe systems at every level of the hierarchy of Figure 1, but with specialization as the systems are more specific at lower levels.

This approach caused us to return to the historical foundations of systems engineering and ask whether its basic ideas could be expressed more precisely. Model-based systems engineering allows us to describe explicitly what had previously been assumed, intuitive, or possibly even miscommunicated. The result of this effort was the *Systematica Metaclasses* (refer to Definitions and Acronyms at the end of this article) [3, 14, 15].

The metaclasses represent the key concepts used by systems engineers, in the form of object classes. This means we can treat intangibles like behavior as if they were tangible objects, and arrange them into classes—a form of object-oriented analysis of general systems. In older fields such as mathematics, this is quite common (for example, mathematical functions, groups, etc.). The Systematica Metaclasses are:

- System
- Interface
- Feature
- Function
- State
- View
- Need
- Domain
- Attribute

Brief word definitions of the metaclasses are listed in the Definitions section. Because this is a formal modeling approach, the above names of the metaclasses are not important [they could be renamed without impact] compared to their *semantic model* (informally represented by the semantic web of Figure 8).

An intuitive feel for some of these is summarized by:

- Systems are *who* can interact (not necessarily humans--anything that can interact)
- States describe *when* interactions happen
- Functions describe *what* the interactions are
- Features describe groups of functions that have *marketable value*

The above list of metaclasses is quite short to claim to cover the complex field of systems engineering. This is illustrated by comparison to efforts to establish standards for systems engineering database information models [19-22], or to the intuitive complexity of large specifications. These more complex examples are specializations of the above metaclasses. The practical measure of success is in real specifications and projects, and in our SE practice we have tested this framework on many systems, technologies, and domains.

Some additions and refinements have occurred and will continue, but the current metaclass framework has become reasonably stable and has been used over years to describe a variety of very complex systems. An example is the large heavy equipment enterprise product line discussed later below. The point in keeping the Metaclass list as short as possible is that we are trying to find the smallest possible set of formal ideas necessary--to productively represent the practical processes and specifications of real world systems engineering in a product line environment.

By keeping this list small, we aim to maximize expressive power and minimize perceived complexity. That the ideas involved should individually be among the most familiar informal systems engineering concepts is no surprise, and aids in their use. Figures 5 and 6 illustrate two examples of graphically oriented Systematica models. These diagrams permit graphical expression in systems engineering versions of UML [16, 17].

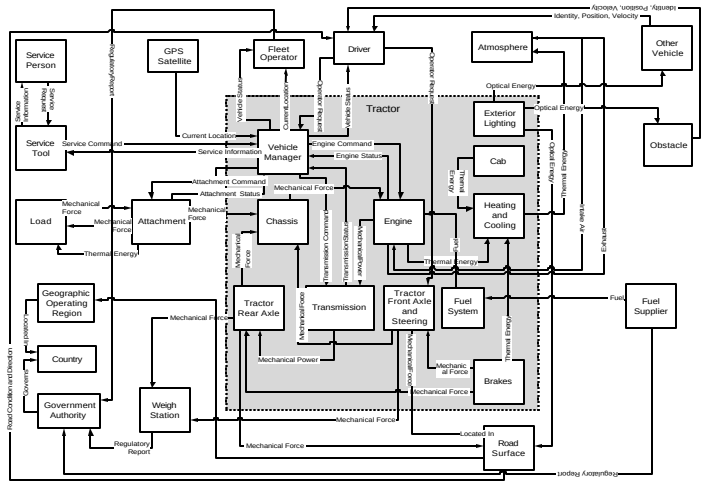

Figure 5: Example Collaboration Diagram

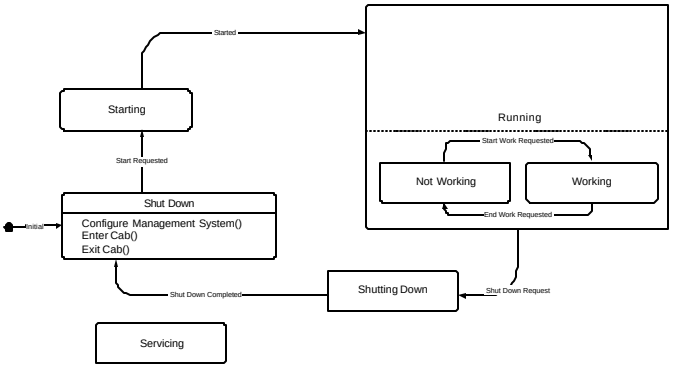

Figure 6: Sample State Machine Diagram

Descriptions of systems using the Metaclasses are inherently *relational*. That is, they consist of specification-like statements made using linked together metaclass objects—possibly in everyday English or other natural language, in mathematical equations, etc. We sought out a relational structure in the first place in order to represent holistic system *patterns* (gestalts) that are common across a family of different systems that varied in other respects.

The inherently *relational* (see Definitions) nature of systems engineering, and of language in general, has been studied at some length as an explanation of the fragmentation that occurs across different technical disciplines and the problem of moving to new technical framework paradigms [23-26, 51]. The idea of reusing or representing patterns formally in specific engineering or other technical disciplines such as building architecture and computer software has had wide exposure [27-29, 35, 36].

Formal models are widely used in engineering, including building blueprints, mechanical drawings, electronic schematics, hydraulic schematics, biochemical metabolic cycles, software class diagrams, etc. They are frequently but not exclusively diagrammatic [5, 7, 9, 12, 13, 16, 17, 32].

The diagrams in each of these cases are inherently *relational* representations. These models also have non-diagrammatic representations (e.g., relational database tables). Systems engineers' diagrams are inherently relational, and engage the visual cognition centers and processes that recognize, analyze, and synthesize patterns [8, 29, 30, 31, 33].

All model-based approaches join together three different worlds, shown in Figure 7:

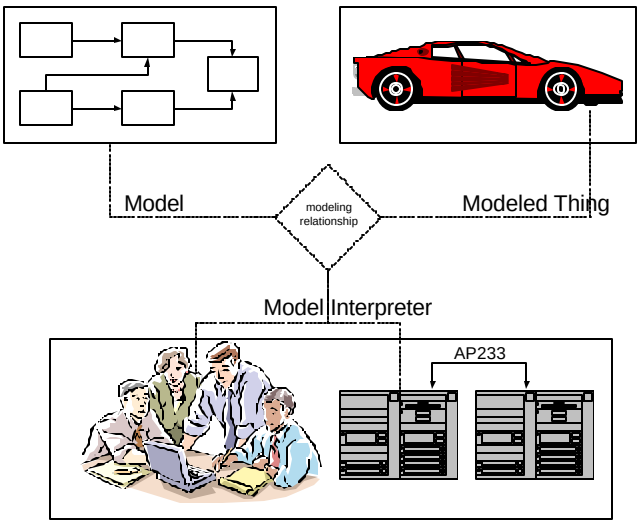

Figure 7: System-Model-Interpreter Diagram

1. The *modeled system* is some real system of interest that we want to describe.
2. The *model* is a separate (information) description of that system.
3. The *model interpreter* is what interprets and acts based on the model.

Model interpreters for engineering models are often human engineers, but not always so. Computer-based tools are frequently used, in engineering, manufacturing, distribution, and service, in which the tool relies on some explicit or assumed semantic data model to describe the system of interest.

This reminds us that current efforts to establish common automated system information models for systems engineering should not be thought of as divorced from establishing clarified formal frameworks for human description of systems [8, 16, 19, 21, 24, 25, 34]—neither should be allowed to become too complex, if it is to be effective. Two efforts in particular can increase the precision of the ideas of traditional systems engineering:

1. Executable Models: One of the ultimate goals of model-based engineering is to obtain "executable" models. This means automated systems in the "interpreter" role of Figure 7 are capable of simulating the operation of the modeled system. They can validate a model early, or evaluate the performance of a system. This is not limited to software based systems, but for any system that is modeled. [34, 38]

2. Interoperable Models: Another of the goals of model-based engineering is for models to be shared between different automated systems. This means that systems engineering, discipline specific engineering, automated test, manufacturing, product data, or other systems can share the same information. [20, 21]

We therefore think of the Systematica Metaclasses as establishing a *semantic web* (Figure 8) which can be viewed as either a clarified formal framework for human thinking and communication about systems, or as an information model for computers exchanging information about systems—the problems are very closely related. The exchange of SE data between machines requires a precise human agreement about its meaning—a goal of standards such as ISO10303 AP233 [20, 21]. The underlying structure of meaning of the Systematica Metaclasses has already allowed us to move systems engineering information across systems engineering tools and information technologies of several suppliers.

What is different about such a model-based approach is that we are not simply parsing specifications into paragraph and sentence objects, re-linked together in dependency, containment, tracing, or derivation relationships. Instead, we are expressing models in the

explicit metaclasses that make up an underlying (now formal) minimal language of systems engineering. This describes what we call the "natural" relationships of systems, shown in Figure 8. They go considerably beyond derivation, dependency, containment, or tracing relationships.

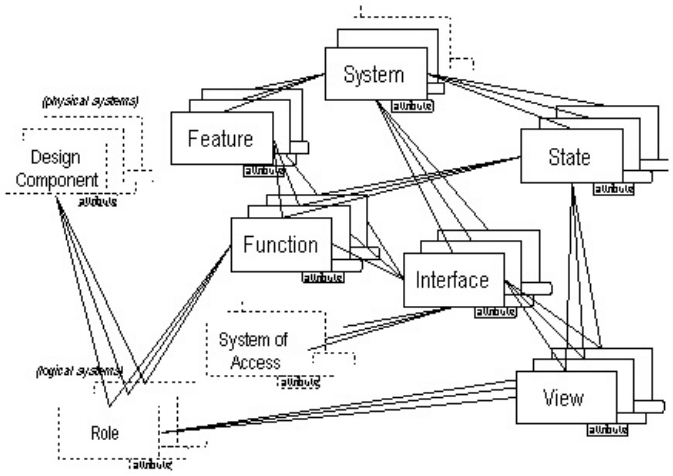

Figure 8: The Semantic Web of Metaclasses

The details of this approach are described elsewhere [3, 15, 18, 37], and are comparable and closely related to contemporary modeling approaches used in specific engineering disciplines or general systems engineering [5, 6, 9, 16, 20, 34, 38, 39].

What is unique in this approach is that it has been established from the beginning to efficiently represent families of systems / product lines.

GENERAL RESULTS OBTAINED, LESSONS LEARNED, AND FINE TUNING

This section summarizes some of the general results we have obtained with this approach, including lessons leading to refinements. Later sections describe results of applying it in a large product line-oriented enterprise.

Expressing Local Standards for Engineering Processes

Instead of proscribing a single rigid process for systems engineering, Systematica provides a framework language (*metamethodology*) for the description of local processes and corporate standards. Systematica metaclasses provide a general model-based environment for expressing and improving SE processes. However, this framework is not totally flexible—or it would not be a framework. It causes us to think and communicate in a structure of the natural relationships of systems, not fluid natural language.

As shown in Figure 9, a specific corporate methodology meeting the requirements of enterprise standards and programs can be described by Systematica. What is different is that it provides a very explicit way of modeling these processes (using Systemtica itself). In addition to supporting challenges such as product line global optimization engineering, this approach also has other benefits associated with clearer models of the engineering process. Many notations have been used to describe engineering processes. In Systematica, we use the same modeling notation to describe engineered systems as we do to describe the engineering process. This can help in the "definition" aspects of CMMI, ISO-9000, 6 Sigma, and other process improvement or systemic programs.

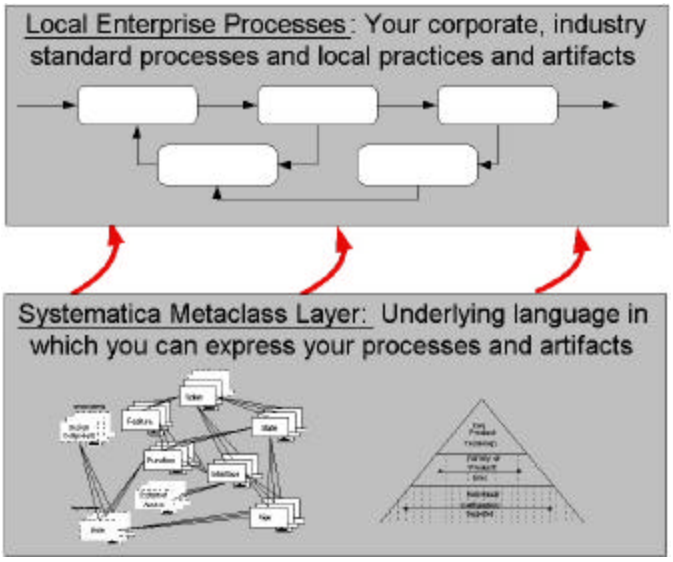

Figure 9: Metaclasses Express Enterprise Process Standards

Configurability of Engineering Processes

Once a local engineering process is described using Systematica, it can also be further *configured*. Systems engineering (SE) provides general principles and practices [1, 4, 6, 9, 16, 19, 37, 40, 42, 43, 44], but there are unique aspects to each project, customer application domain, or project team [41].

We have learned that Systematica enables us to configure the systems engineering process at the beginning of a project. This is important in order to align the project methodology with the customer, project success model, engineered system, budget, and schedule [41]. Since Systematica provides a Metaclass based language but does not force a specific process, some of the things we configure to the project are SE artifacts (documents or views), assignment of SE roles to specific people, organizations, and tools, process gate exit criteria, and degree of process concurrency or

sequence. We can configure "light" SE processes this way, when they are appropriate.

A High Return on Investment "Sweet Spot"

While model-based systems engineering has many benefits, we have learned that one "sweet spot" consistently produces high returns for limited investment and low risk. Systematica describes systems using various levels of models. Among the SE processes are the High Level Requirements (HLR) Process and the High Level Design (HLD) Process that create the similarly named models.

The HLR process produces only the top level framework of summary requirements specifications for a system or family of systems. The HLD process produces only the top level design architecture summary for those systems. Requirements (whether primary or derived) are clearly differentiated from designs in this approach, by their visibility to external systems. Other Systematica processes produce detailed level specifications and other artifacts.

The HLR and HLD specifications are generally relatively modest amounts of information (e.g., often 10-50 pages in document form for even complex systems) and are frequently produced in relatively short times (e.g., often 2 days to 2 weeks for complex systems) by experienced systems engineers or domain experts. Nevertheless, we have repeatedly seen that these basic artifacts can have very high value, including:

1. Rapidly creating and communicating a shared understanding of the overall structure of system requirements, prior to detailing.

2. Gaining a rapid discovery audit for missing requirements, by actor, feature, and situation.

3. Creating a project management framework for organizing specification writing as a construction process, with measurable milestones and completeness metrics to the level of individual functions. This connects project management to system engineers.

4. Creating a high level mapping of market and customer expected features, in the language of the market or user, to the engineering specified system functions, in the language of the engineer—all while still at the high level requirements stage. This establishes an ongoing connection between marketing and engineering.

5. Rapidly creating, communicating, and validating a shared understanding of high level design (system architecture) and the allocation of required functional roles to design components owned by individuals.

6. Negotiation of complex features across multiple individual owners of participating subsystems.

7. Organizing the plan and work of generating test cases, across the high level situationally-based requirements framework.

8. Validation tracing of informal Needs to individual functions before detailed specification work begins.

9. Explicit representation of domain knowledge that is otherwise often in the heads of subject matter experts, but not readily available to the other members of project teams and their supporters.

Our experience has been that companies producing large amounts of specification detail documents frequently do not generate these simple frameworks, which can have very powerful organizing effect on the work of others working at more detailed levels. In the absence of these frameworks, detailed specifications can seem overly complex and are less effective, and outcomes are at greater risk and more difficult to monitor and manage.

The Power of Logical Systems

Systems engineers often work hard to abstract and separate requirements from physical designs, to clarify real needs or to improve migration of IP across future technologies. Partitioning logical systems can seem an overly abstract exercise when we try to share the results with others.

Systematica encourages us to explicitly model both logical systems (functional roles) and physical systems, and to explicitly show allocations. Since this can be done with intuitive diagrams (e.g., Figure 5), improved communication about logical versus physical systems is obtained, and more people learn to partition and analyze logical architecture.

Scope and Sustainable Innovation: Incremental Payback for Incremental Investment

The power of using repeatable patterns in modeling complex families and processes is infectious. Once it is learned, there is an upside risk of taking "too big a bite" in planning the scope of what to model. It is not necessary to model the entire product line of the enterprise, or all its internal processes and systems, to obtain paybacks.

We call modeling system families, on an incremental basis, Uncovering the Pattern™. The process of building models can be thought of as uncovering an enterprise-wide pattern one region at a time.

We have found that it is possible to plan incremental Uncover the Pattern / Harvest the Pattern cycles, with each covering another area of the enterprise, its products, or its systems. Each of these cycles can harvest benefits,

such as reduced costs or time to market. Any single cycle can focus on a single product, product line, internal system, application, or operational site. Benefits harvested in one cycle can be used to help fund the next cycle.

Specifications as a Model Construction Process Enhances Measurement and Accountability

In earlier days of manufacturing, common content of products was interpreted to mean common physical parts. In today's more complex systems, common content may be high without common physical parts—even for mechanical systems. Today's system products can embody the results of high cost investment in company learning curves of design, production, or service of products that have high similarity of other kinds.

For example, approaches to fuel injection control, suspension system dynamics, fault tolerant behavior, or visual styling of similar products may embody a high degree of common content, but not in the form of common physical parts. How do we represent, in model blueprint form, the common abstract ideas of vehicle suspension, aircraft landing, or operator environment, without being specific to a certain model of product or type of system--so that they can be configured across different specific products? An excellent further example suggested by a reviewer is the impact of common computer instruction set architecture (e.g., IBM System/360™, Intel 8080™, etc.) on global economics—all without involving common physical parts.

The metaclasses permit this common content (both requirements and design) to be abstracted into high level classes that are reused through inheritance by the specific product models of a common product line. This allows the product lines to inherit what is common to all of them from higher level systems of Figure 1.

These members of the metaclasses are objects that have attributes. We can count the number of functions, interfaces, features, design components, or other objects. We can analyze the values of their attributes, such as completeness, approval levels, cost, weight, or other aspects. This means that we can support engineers' intuition (still highly valued) with metrics that measure common content, completeness, reuse, and other aspects of product content, even though it is not physical. This means we can begin to account for and manage globally the real intellectual assets of the enterprise.

We have discovered a practical set of Gestalt Rules™ [2] that permit straightforward measurement of the degree to which a product or system conforms to the overall pattern rules set up for its product line, corporate architecture, or other pattern. This is equivalent to moving from intuitive finance without bookkeeping to use of actual accounting—intuition is still of great importance, but it can be shared with others that are not architects (or accountants). This makes it possible to formalize the management of two cycles of global product pattern management in the enterprise (see Figure 10):

1. Uncovering Patterns: Emerging patterns in individual systems, products or market segments can be recognized, generalized, and pulled up into product line or global enterprise assets, for future harvesting use in other systems. These may be domain specific patterns (e.g., engine control patterns, defense related patterns, etc.) or may be of a more generic nature (as in the next section)—but they should be economically useful, not just esoteric.

2. Harvesting Patterns: Existing pattern assets for product lines or corporate standards can be rapidly "harvested" by bringing them into new or improved products or systems, creating new market value, revenue, or savings.

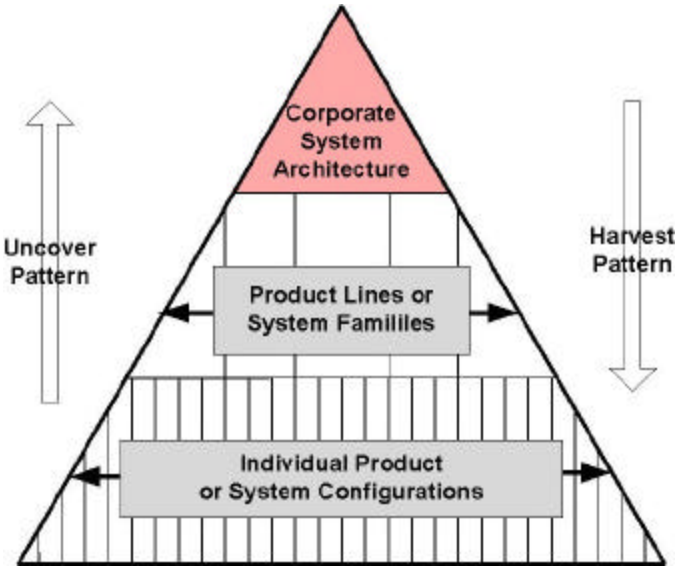

Figure 10: Uncover/Harvest Pattern Cycle

The Embedded Intelligence Pattern

One of the more important upper level abstract patterns to emerge from this process is the Embedded Intelligence (EI) Pattern. This describes a related set of ideas that must be present any time that intelligence (whether in the form of information technology, humans, or other forms) is embedded in systems. Figure 11 illustrates this pattern in the form of four types of logical systems (roles) that emerge in all such systems; each of these roles is defined in the Definitions.

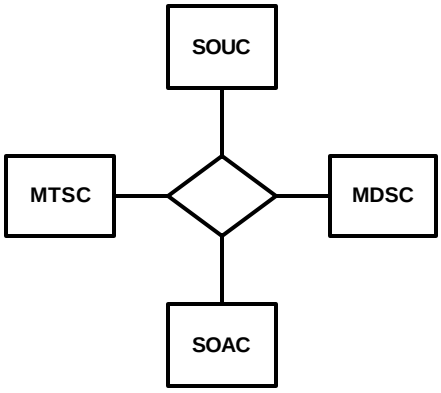

Figure 11: The Embedded Intelligence Pattern

The MTSC embedded logical management role of Figure 11 is further specialized into five subclasses of embedded intelligence (system management), called the system management functional areas (SMFAs):

- Performance Management
- Configuration Management
- Fault Management
- Security Management
- Accounting Management

These are defined in the Definitions section.

The Embedded Intelligence Pattern has been applied in many kinds of domains, including guidance systems, heavy equipment and automotive embedded controls, telecommunications network management, business processes and human organizations, and others [3]. A report on results of its use across the product lines of one large enterprise is included later in this article.

The Communications Pattern

Another important abstract pattern that has emerged out of this process is the Communications Pattern, which applies to datalinks in commercial vehicles and defense systems, to sensors, actuators, and specialized electronic signaling circuits, to human-machine interface technologies, to hospital or factory instrumentation and control systems, and to inter-personal communications processes.

This pattern is particularly based upon Systems of Access and Views, both listed in the Definitions. It permits the design of application product families that can migrate faster and at lower cost over different technologies in the future.

Relationship of Methodology to Automated Tools: Horse before the Cart

Our belief is that methodology comes first, automated tools second. By this we mean that organizations must decide what process they are following or want to improve before trying to automate aspects of it, and that selecting an automated tool [38-39, 46-49] first is not an effective way to decide what the target process to be automated should be.

An earlier section above described the modeling and configuration of a systems engineering process fit to the situation, using a meta-methodology such as Systematica. Once this is done, it is frequently helpful to be able to automate parts of it with appropriate tools, and also helpful if these tools are already available in the corporate standards asset set.

General tool suppliers may be subject to economic forces to maximize their potential market by not advocating a single systems engineering methodology. The result of this may sometimes appear to be a lack of a specific approach, or a high cost of set-up of a generic tool for a specific company's processes in a new project. One of the purposes of the Systematica Metaclasses is to allow systems engineering information to be based upon common underlying semantics so that it can be shared across tools. This is a goal of ISO10303 AP233 [20-22]. It is an enabler of both tool users and tool suppliers.

We have arranged, for example, for the Systematica Metaclasses to reside in tools based on the products of three well-known SE tool/information technology suppliers. The open inter-system exchange of this metaclass data in the form of XML is a further aid to the open nature of the resulting infrastructure supporting a Systematica Methodology – based process. This allows us to establish the Horse (process) before detailing the Cart (tools).

Documents, Databases, and New Types of Views

Systems engineering processes use and produce *information*. Many efforts have been undertaken to create computer-based databases of systems engineering information. Even though the resulting engineering process may be called "data base driven" instead of the more traditional "document driven", there is still an expectation today in most organizations as to the availability of specification documents.

Systematica as a methodology considers all documents and other artifacts as *views* of the underlying base of SE information, whether stored in a computerized database or otherwise. The specific form (e.g., outline, format) of these documents is not the same as the engineering processes, but is an important part of the configuration of the process. By using locally-specified templates that describe these forms, it becomes possible to automatically generate these documents, or other on-line

views (e.g., web-based, etc.) in different forms suited to different users, while maintaining a common base of data "behind" them, so that they are consistent.

Models are less ambiguous and more efficient representations than prose (e.g., consider the prose required to describe an electronic circuit schematic drawing). However, we do not intend for models to appear foreign to specification readers. An important goal is that model-based specifications should *look like and read like "good specifications"*. Just because we are using underlying model-based approaches to describe systems does not mean that we should expect engineers to accept less than high quality engineering specifications.

If anything, model based systems engineering should produce information that is even more acceptable to engineers, not information that is strange or harder to use than other approaches. This also applies to non-engineering users of the same underlying data, who need appropriately structured views of information for use in Marketing, Purchasing, Materials Management, and other areas.

Since metaclass information can be viewed as objects, we can not only generate traditional specification documents from them, but also new forms of information. Figure 12 below illustrates one such construct—the System Model Tree. Particularly suited to use on tools that use Systematica Metaclasses, this view allows an engineer to view a tree structure summary of no only physical but also *behavioral*, or other aspects of a system or product line, and to graphically drag and drop (manipulate) product features, functions, interfaces, subsystems, and other aspects, to configure systems.

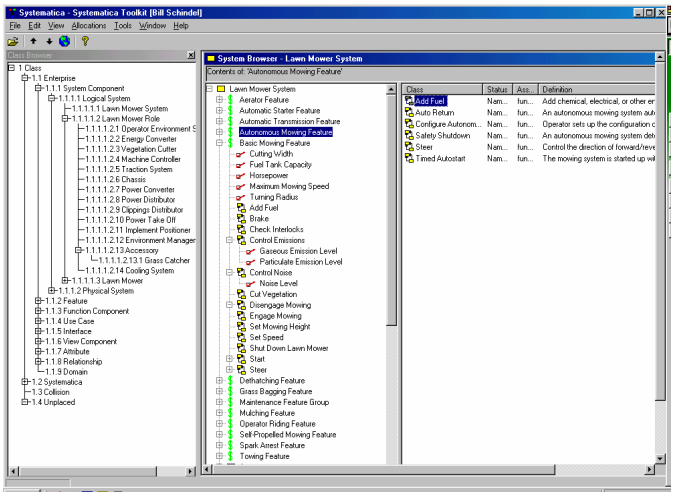

Figure 12: Browsing the System Tree: An "X-Ray" of the Anatomy of a System

How Much to Model

A lesson learned by many new to formal modeling is not to model too much—most new modelers tend to over-model. That a correct fact can be added to a model does not mean that it is a useful addition, or worth the complexity it adds. We should begin by remembering that all models are approximations of reality anyway, and ask what will be most useful.

"All models are wrong—some models are useful," wrote G. E. P. Box [45]. By providing a limited set of metaclasses, we have been able to reduce the complexity of resulting models, but there is still a need for skill of modelers in focusing on what is important. Proliferation of too many types or components is not desirable, and can be measured, evaluated, and managed.

The Value of Reference to Environmental States

Modeling languages such as UML encourage the use of formal states in modeling. State machine models are long accepted in electrical engineering, software engineering, and some other fields. [5, 17, 24] This has been most common in *design* models.

Systematica encourages us to also consider state models of *requirements*. Only a few disciplines, such as real time system design, have strongly encouraged this in the past. However, the methodology we are describing tells us that for all requirements statements, "*when* does this requirement apply?" is a reasonable question to ask and specify. Environmental state models are models of the external environment of a subject system. They describe how situations can unfold on a temporal basis in the environment of a system.

Two references to methodologies can be cited that are related to this idea:

- The use case method in software engineering [13, 32] identifies things called use cases, which are environmental situations that must be satisfied by systems, as a requirements specification and analysis method. However, software engineers sometimes object that use cases are not states. Indeed, they are most frequently not represented as nodes of state machines; they are also about states of the environment of a system, not necessarily internal design states of the system. Thinking about iterating decomposition of system layers, however, reveals that one system's design is another system's requirement.

- The use of "concept of operations", "operational views", or "scenarios" has been shown to be a powerful way to elicit or communicate requirements concepts in the perspective of the stakeholder. These may be seen as sequences that are particular paths selected from (typically many other) paths through the state machine model. These typically display interface level views over the scenario. [19]

Figure 6 illustrates a simple state machine for the environmental states of a system. More complex models may take advantage of concurrent states and sub-states.

Benefits to Global Enterprise versus Individuals

A key motivation in developing this approach was the ability to *globally* optimize competitiveness of whole enterprises, as summarized by Figure 13. We have learned that it is equally important to assure that individual engineering practitioners enjoy a perceived *individual* benefit in their work. Figure 13 illustrates the idea that an engineering methodology should bring benefits to both the enterprise as a whole (as perceived by its senior management) and to the individual engineering practitioners within that enterprise.

The approach described here permits individual product line architects to have greater impact across more product models. It encourages the discovery, expression, and propagation of elegant architectures, simplifications, and powerful abstractions, and directly measures their occurrence. The organizational culture, processes, and tools should all combine to reinforce these benefits in both dimensions.

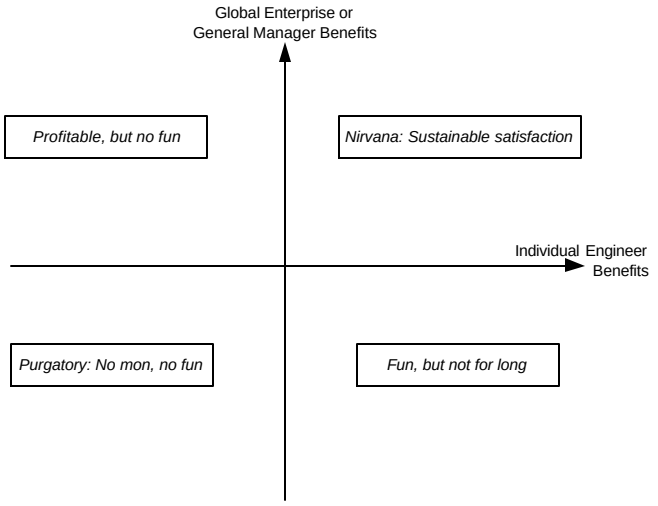

Figure 13: Meeting Enterprise and Individual Interests

APPLICATION TO A MAJOR PRODUCT LINE ENTERPRISE: CATERPILLAR E&ES

Caterpillar Inc. is one of the world's largest suppliers of heavy equipment and engines, across global markets for specialized machines, machine power, and machine subsystems, for construction, mining, agriculture, off road and on road vehicles, industrial, marine, power generation and other markets. The company's product lines have become increasingly sophisticated and complex, with diversity across different applications, types, and market segments. This sophistication and complexity are further enabled by Caterpillar's use of embedded electronic intelligence and communications technology in these products and in the company, customer, and partner tools and processes that support them.

AN ELECTRONIC CONTROLS PRODUCT LINE

Illustrated by the example of Figure 14, Caterpillar's Electrical and Electronic Systems (E&ES) business unit provides product lines of embedded electronic controls for all Caterpillar machine and engine products. This is made up of component technologies that include rugged electronic control modules (ECMs), embedded real time control software, sensors, actuators, operator interfaces, on-board harness and networking, and off-board wireless and other forms of networking to a variety of service and management systems. In addition to component technologies, Caterpillar E&ES makes extensive use of systems engineering throughout the life cycle of these systems products, and as a part of the larger systems engineering effort of the corporation as a whole. These efforts include continuous improvement of the engineering process itself, as well as the product systems offerings [50].

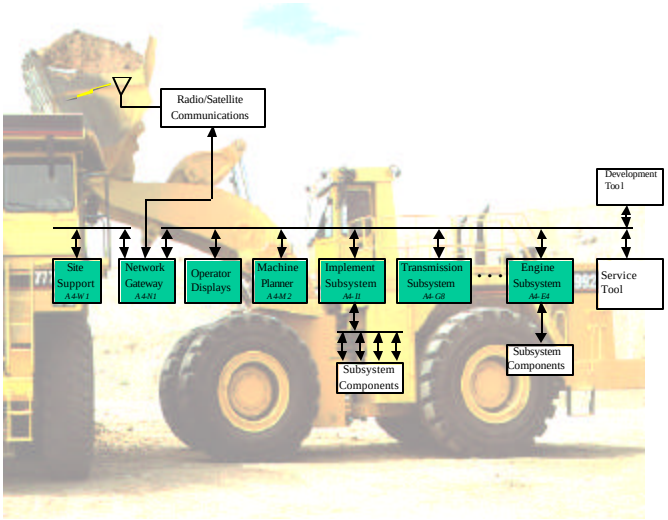

Figure 14: Networked Machine Control Systems

Electronic controls at Caterpillar began for transmissions in the early 1980s, spread to engine emissions and performance management by 1986, later to implement, braking, and other machine subsystems, and in recent years have been the subject of machine/vehicle level integration. ECM hardware technology for E&ES is currently entering its sixth generation. Spread over approximately 200 Caterpillar and third party machine models, there are approximately 1,800 unique E&ES

production subassemblies representing product line configurations currently recognized. Another measure of complexity is the approximately 7,000 data link message types supported across a number of data link types.

E&ES also provides for integration of its embedded on-board products with off-board portable and fixed systems provided by E&ES, other Caterpillar units and suppliers, dealers, and end customers. This includes the integration of wireless and other forms of data networking, further increasing the scope and complexity of the total fleet level systems that result.

These factors combine to make Caterpillar E&ES an outstanding candidate for continuing emphasis on product line systems engineering and complexity management. These further improve global product line capability for end customers and applications, minimize cost by sharing assets, and better manage complexity.

AN INCREMENTAL APPROACH TO SYSTEMS ENGINEERING PROGRESS

E&ES has pursued an adaptation of Systematica Methodology incrementally since first educational exposure to it in 1998. Configured and adapted to Caterpillar's New Product Introduction (NPI) Process, along with the needs and processes of the company and its engineering teams, an E&ES version of this methodology has been in selective use for several years. The refinement of this process is still underway. The following sections report on the initial results we have begun obtaining in the several years to date.

INITIAL RESULTS OF THE PRODUCT LINE SYSTEMS ENGINEERING METHODOLOGY IN AN ENTERPRISE

A Product Line-Wide Logical Reference Architecture

Shortly after adopting the systems engineering methodology described here, we used its differentiation of logical and physical architecture to establish a common logical architecture with scope covering substantially all of E&ES hardware and software products for on-board and off-board applications. This represented an important step in comparison to past hardware-focused architectural representations, and has improved the framework we use to organize and manage intellectual assets for improved portability over changing physical technologies. Although we continue to refine and detail this architecture, it has held up well as we have applied it to various machine, machine subsystem, and networked systems engineering projects as a unified reference architecture.

This logical architecture covers four levels of product decomposition (logical systems containment hierarchy):

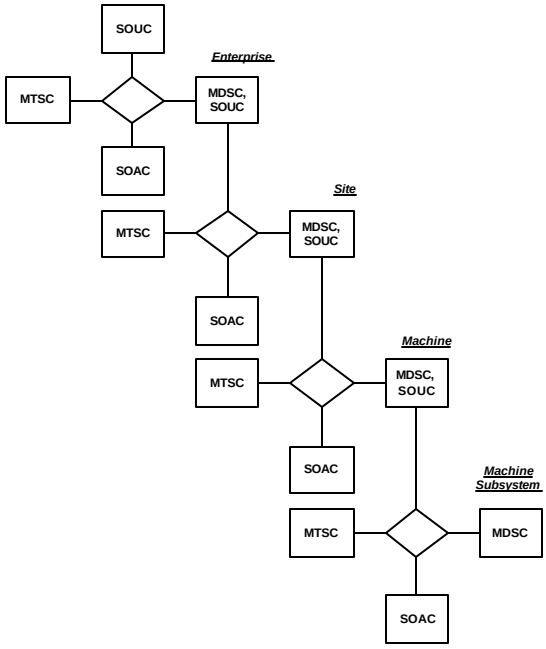

Figure 15: Logical Containment Hierarchy

The same logical architecture also covers six levels of product specialization (logical systems class hierarchy), as shown in Figure 16.

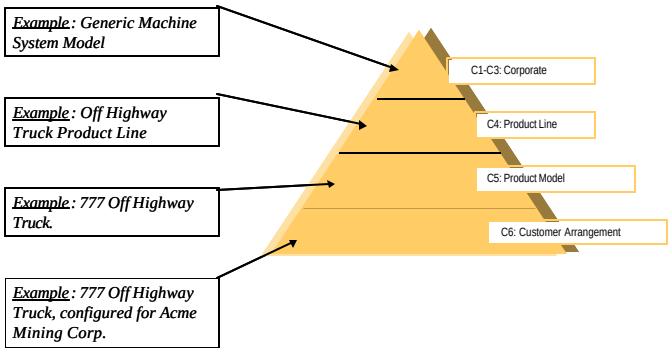

Figure 16: Logical Class Hierarchy

Another view of the integration of the E&ES site control, machine control, machine subsystem control, and common resources is summarized by the wheel of Figure 17.

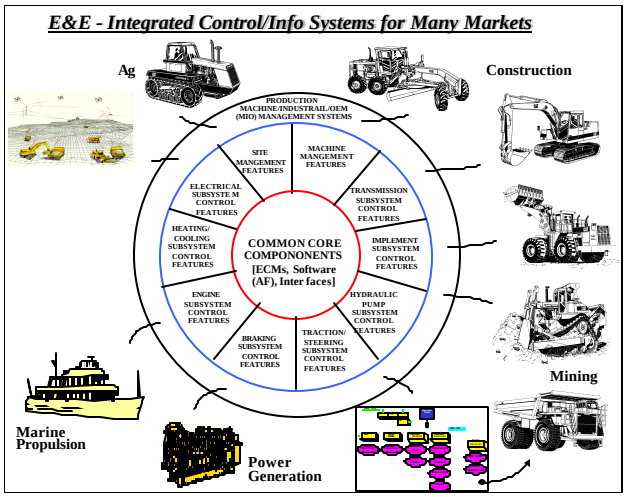

Figure 17: Common Assets Across Many Products

The implications of this architecture continue to unfold. Most recently, E&ES has established an Architecture Control Board of senior engineering experts who preside over its overall structure. This board has begun using the logical partitions (Figure 18) resulting from this architecture to manage the intellectual assets of the product line across a variety of engineering tools, systems, and repositories, and expects to see it continue to migrate across other information management systems and processes. Individual domain owners are being given responsibility for each of the machine and machine subsystem domains of the architecture, integrated by a common systems engineering process and architecture. An E&ES Systems Standards Team has been in place for several years, providing additional support for these standards.

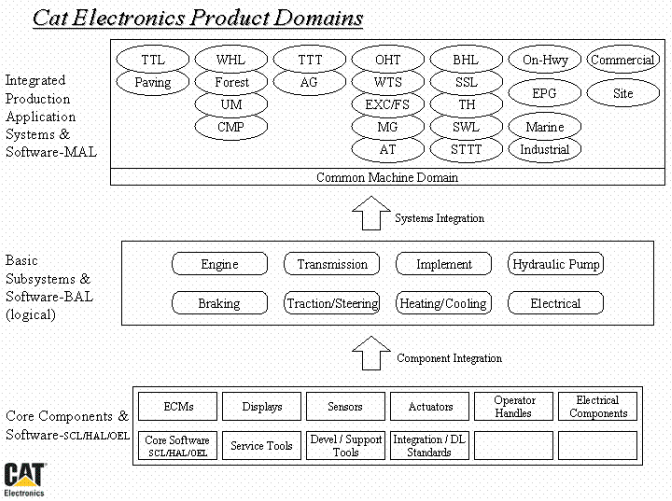

Figure 18: Product Line Domains

Improved Product Development Gates and Metrics

In configuring and adapting this methodology for Caterpillar use, E&ES has most recently been establishing a series of process stages and completion metrics to support them. Embedded within the Caterpillar NPI Process, E&ES views these stages as a series of process "gates" (C, D, E, F, R, S) spread across the traditional systems engineering decomposition and integration "V" diagram [4] of Figure 19.

Because of the explicit metaclass models of the Systematica methodology, E&ES has been establishing a set of computer-supported metrics to help better quantify the status of an engineering project. Traditional measures of completeness of a Systems Requirements Specification (SRS), for example, rely on relatively subjective judgments about such a document. By using metrics based on the status of each object in the specification, we can more objectively and in more detail judge the current status and content maturity of a specification, and focus attention where it is most needed.

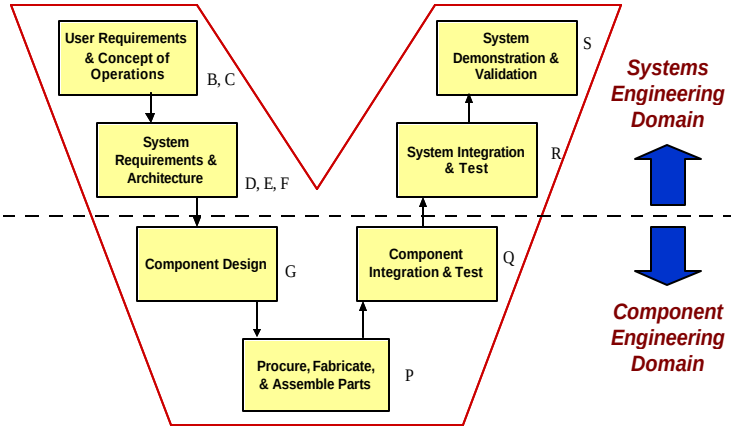

Figure 19: Traditional SE Process "V" Diagram [4]

Implementation of Corporate Standard Processes

Caterpillar and Cat E&ES have adopted or developed standards which include systems engineering aspects, among these the Caterpillar New Product Introduction (NPI) Process. In addition, other programs such as ISO-9000, 6 Sigma, and discipline specific standards and processes such as E&ES (Level 2 CMM) software engineering are supported by the systems engineering process.

These standard processes often demand certain results of the systems engineering process, without describing a specific practice to accomplish them. The methodology we are adapting provides a productive means of actually implementing on a practical basis the kinds of systems engineering processes that these standards call for—and does so while providing a product line-oriented approach that our business requires.

Managing Requirements as Assets

It is common to hear of managing *design* data to increase re-use of design components. This methodology also permits us to manage *requirements* as Intellectual Property assets, using model-based representations of requirements, beyond prose alone. We achieve re-use of design assets by first managing the re-use of modeled patterns of requirements, in the form of logical systems, features, functions, and interfaces, and this in turn leads to re-use of designs. More important, the class hierarchy of Figure 1 means that we extract from our various products the *portion* of their content (requirements and design) which is shared in common, whether it is large or small, and carry it to higher layers of the product class hierarchy.

We also improve our retention of domain expertise by managing requirements as assets in the form of these explicit models.

For example, two different electronic engine products may behave differently, but still have certain aspects (engine control patterns) that we can manage as a common set of requirements and design assets. Even though the different engine control systems require different configuration for their ultimate use, we can still save by managing the common portion of engine behavior as a unitary asset. The abstraction required to accomplish this begins with requirements, and only later is the same done for design of electronic software and hardware.

Organizational Impact and Productivity

Maintaining assets at different levels of Figure 1, and managing the continuing evolution process of Figures 2 and 10, mean that we have separate, coordinated "blueprints" (models) at each of these levels. This is a different way of doing product line business than simply cloning end product specifications and moving people, as shown in Figure 2.

One impact of this approach is that we establish "owners" of these assets, at different levels of Figure 1. This means that individuals are responsible for higher level abstract or generic domain assets, and provide the necessary coordinating influence across product lines.

This might seem like more effort than would be required for a single product—and it would be, in a single-product business. However, Caterpillar is not a single-product company, and we are organizing our work around this hierarchy because it is globally less effort to coordinate products in this way than to treat them as independent, with separate engineering staffs.

Benefits to Individuals

By providing a more natural structure relating product line framework assets to individual products, this approach makes it possible for the ideas of individual engineers to have wider impact across product lines. Ownership and responsibility for product line patterns versus individual products are more clearly delineated, increasing satisfaction and lowering the frustration of miscommunication. Treating requirements as models allows engineers to use common language for system requirements and system design, and to have clearer ways to communicate with others.

Automated Specification Generation

The use of System Requirements Specification (SRS) documents has been well established in E&ES for some years, although the precise interpretation of their structure, content, and completeness has not always been uniform across the whole organization. Issuing word processor-based outline oriented templates for these was of some help, but still did not offer uniform interpretation of what should appear (or when) in the various sections of a prose-oriented SRS to be acceptable. The arrival of model-based systems engineering and supporting automation has allowed E&ES to advance this area.

Building on earlier work, in 2001 we began more automatic generation of some of our E&ES SRS documents, using tools that produce these from model-based databases. We have been building up databases of metaclass-based requirements and designs in DOORS™ and Systematica™ Toolkit (Microsoft Windows™ based) integrated with it, and have been using Systematica Toolkit to generate SRS documents from this model database automatically, once the properties of a subject system have been selected.

This approach allows us to establish document templates that are used to specify the structure of information that will be automatically inserted into a document, from the model database. One of the first SRS documents we generated in this way was a compact "handbook" publication that specifies our overall E&ES architecture at upper levels of Figure 16. We are now working on templates to automate the generation of product tests based upon requirements.

Controls and the Embedded Intelligence Model

The central concept of Caterpillar's E&ES product offerings is embedded electronic controls, although the specific technologies used for processors, software, sensors, actuators, and networks varies over products and time. We have been building up a common requirements model across all of these using logical systems of the Embedded Intelligence Model of Figure 11 and the SMFAs. There is a great simplifying advantage in being able to view the broad range of Caterpillar's many applications through the lens of this unifying abstraction.

This abstraction has allowed us to cover a very large range of varying requirements with a common logical model of requirements, and to separate it from the physical hardware and software technologies used to

implement designs. This separation provides improved leverage to migrate common requirements over a range of technologies for different product configurations or new technology introductions, without having to untangle requirements from previous design technologies through reverse engineering for each new generation.

We have further applied this model to manage the migration of roles from human operators to automated machine controls. Our products have followed the general trends seen earlier in avionics (involving skilled pilots) and manufacturing machine tools (involving skilled machinists), as embedded controls have taken work load off the human operator over generations of products—in the extreme case resulting in autonomous systems that operate with little or no local human intervention. The Systematica Embedded Intelligence Model allows us to treat human management and electronic management (controls) through the same unified MTSC abstraction. This is allowing us to migrate required roles between human operators (read training manuals) and electronics (read software).

Integrated Datalink and Circuit I/O Model

Cat electronic control modules (ECMs) communicate with each other and certain other equipment using either the Cat Data Link (CDL) protocol, or any of several different standards based protocols, such as SAE J1939 or J1587. There is an accelerating pace of evolution of these data links as key integrating technologies for E&ES systems products and the larger systems into which they are integrated.

Cat ECMs also communicate with the machine subsystems they monitor and control by specialized circuits connected to embedded sensors (e.g., pressure and temperature sensors), embedded actuators (e.g., solenoids), and human interface devices (e.g., operator panel displays, joysticks, pedals). These circuits are frequently not data links, and may utilize pulse width modulation (PWM) or other specialized electronic circuitry.

A large machine may have hundreds of such control circuits, including data links and many instances of several types of sensor and actuator circuits and devices. The overall systems engineering process covering all of these across all Caterpillar machines is a very complex task.

The Systems of Access (SOA) portion of the Embedded Intelligence Model of Figure 11 provides us with a major unifying construct to dealing with this disparity in a structured way. It allows us to migrate common application requirements across different types and generations of data links and physical sensor and actuator circuits, plus human interfaces, with considerably less apparent complexity, through a unified treatment.

Portable SE Information Base

We have been accumulating a common central information base of requirements and design information in a common repository based on the metaclass model summarized by Figure 8. This information has been used in more than one type of engineering tool within E&ES, and we are looking forward to migrating it to other tools using the XML based open data exchange interface supported by the Systematica Methodology. We also process this data in simpler tools such as spreadsheets, as well as through more sophisticated desk top tools that convert system drawings into metaclass data.

The ready availability and portability of this information across current and future automated tools is almost as important to us as the conceptual value of the metaclasses to unify thinking and communication across people. In combination, the Systematica Metaclasses provide a simple and elegant foundation upon which we have built a more complex structure ranging across the many products of E&ES.

FUTURE PROCESS IMPROVEMENT

Although we are beginning to realize the benefits summarized above, we feel that in many ways we are still in the early stages of system engineering in this new approach, and see more potential ahead as we exploit and harvest more of the benefits of the full vision. Some of the most important work we have underway includes:

1. Involving more logical Domain Owners and the Architecture Control Board in delegated ownership responsibilities

2. Increasing access to the common database for more systems engineers through a seamless client-server model

3. Propagating the system information structure and assets to more interoperating divisional tools and information systems (e.g., simulation, project management, software development, test, configuration management, etc.)

4. Continuing to populate the database as we specify new systems products

5. Training additional participants in the process and the modeling approach

6. Refining the E&ES configuration of the methodology, including expanded SRS templates

7. Fuller integration of processes and tools, such as inclusion of model artifacts in the formal NPI process

8. Gaining increased leverage from related standards (e.g., OMG UML, ISO 10303-AP233)

9. Support for CMMI assessment and improvement of system engineering processes

10. Incorporation of executable models (for system verification) and model driven automatic code generation (for increased development productivity) into the systems engineering process

CONCLUSION

By building upon a formally defined foundation for systems engineering information, the model-based systems engineering methodology reported on here is extending the traditional benefits of good systems engineering practice into the environment of global product lines and families of systems. This approach delivers short-term incremental benefits for incremental investments in modeling, while positioning the enterprise for the more complex and competitive environments of the future.

ACKNOWLEDGMENTS

The authors acknowledge the extensive contributions by the Caterpillar and ICTT engineering teams to the progress reported in this paper. The authors are also grateful to SAE reviewer Eric Swenson for his suggestions for improvements to this paper.

REFERENCES

1. *Capability Maturity Model™ Integration (CMMI™), Version 1.1: CMMI for Systems Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing*, Carnegie Mellon Software Engineering Institute, 2002.

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

3. Schindel, William D., and Rogers, Gloria R., “Methodologies and Tools Supporting Continuous Improvement”, *J. of Universal Computer Science*, March 2000.

4. Arunski, Karl; Martin, James; Brown, Phil; Buede, Dennis, *Systems Engineering Overview*, Presented to the Texas Board of Professional Engineers, September 10, 1999.

5. Booch, Grady, *Object-Oriented Analysis and Design With Applications*, Second Edition, Benjamin Cummings Publishers, New York, NY, 1994.

6. Buede, Dennis M., *The Engineering Design of Systems: Models and Methods,* John Wiley & Sons, 1999.

7. Chen, Peter, “The Entity-Relationship Model: Toward a Unified View of Data*”, ACM Trans. On Database Systems* 1, 1(March), 9-36, 1976.

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

9. Oliver, D.W., Kelliher, T.P., and Keegan, J.G., Jr., *Engineering Complex Systems with Models and Objects*, McGraw-Hill, 1997.

10. Rosen, Robert, *Anticipatory Systems: Philosophical, Mathematical & Methodological Foundations*, Elmsford, NY: Pergamon Press, 1985.

11. Rosen, Robert, *Life Itself: A Comprehensive Inquiry into the Nature, Origin, and Fabrication of Life*, New York: Columbia University Press, 1991.

12. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W., *Object Oriented Modeling and Design*, Prentice-Hall, New York, NY, 1991.

13. Rumbaugh, James; Jacobson, Ivar; Booch, Grady, *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1999.

14. Schindel, William D., “Does Our SE House Have a Foundation?”, presentation to International Council On Systems Engineering, May 22, 2002.

15. INCOSE Crossroads of America Chapter of International Council on Systems Engineering (INCOSE), special interest group on Foundations of Systems Engineering, http://www.incose-coa.org/disc2_welc.htm

16. Systems Engineering Domain Special Interest Group (SE DSIG) –home page in OMG web site: http://syseng.omg.org/index.htm

17. Web site for OMG Unified Modeling Language (UML)--http://www.omg.org/uml/

18. Web site for ICTT Systematica--http://www.ictt.com/systematica/index.html

19. US Department of Defense, *C4ISR Architecture Framework*, US DoD Joint Architecture Working Group, Version 2.0, Dec 18, 1997

20. ISO TC 184/SC4/WG3 N911 *ISO/WD5.1 10303-233 Industrial Automation Systems and Integration: Product Data Representation and Exchange: System Engineering and Desig*n, working draft 5, October 30, 2000.

21. *ISO 10303-1:1994-- Industrial Automation Systems and Integration--Product Data Representation and Exchange -- Part 1: Overview and Fundamental Principles,* ISO TC184/SC4 and ISO Central Secretariat, Geneva, 1994/95.

22. Web site for ISO 10303-STEP-AP233 Systems Engineering Data Representation Working Group--
http://www.sedres.com/ap233/sedres_iso_home.html

23. Kuhn, Thomas S., *The Road Since Structure: Philosophical Essays, 1970-1993*, Chicago: U. of Chicago Press, 2000.

24. Schindel, William D., "The Tower of Babel: Language and Meaning in System Engineering", SAE Technical Paper 973217, November, 1997.

25. Carroll, J. B., ed., *Language, Thought, and Reality: The Selected Writings of Benjamin Lee Whorf*, MIT Press, Cambridge, MA, 1956.

26. Hayakawa, S. I., *Language In Thought And Action*, fifth edition, Harcourt Brace Jovanovich, Publishers, Orlando, FL, 1990.

27. Alexander, Christopher, *A Timeless Way of Building*, Oxford U. Press, New York, NY, 1979

28. Alexander, Christopher, et al, *A Pattern Language: Towns, Buildings, Construction*, Oxford U. Press, Oxford, New York, NY, 1977.

29. Gamma, E., Helm, R., Johnson, R. , and Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, New York, NY, 1995.

30. Arnheim, Rudolf, *Visual Thinking*, U. of California Press, Berkeley, CA, 1969.

31. Ferguson, Eugene S., *Engineering and the Mind's Eye*, MIT Press, Cambridge, MA, 1992.

32. Jacobson, I., Christenson, M., Jonsson, P., Overgaard, G., *Object-Oriented Software Engineering: A Use-Case Driven Approach*, Addison-Wesley Publishers, New York, NY, 1993.

33. Miller, Arthur I., *Imagery In Scientific Thought: Creating Twentieth Century Physics*, MIT Press, Cambridge, MA, 1986.

34. Mellor, Stephen J., and Balcer, Marc J., *Executable UML: A Foundation for Model-Driven Architecture*, Addison-Wesley, 2002.

35. Alexander, Christopher, *Notes on the Synthesis of Form*, Harvard U. Press, Cambridge, MA, 1964.

36. Shaw, Mary and Garlan, David, *Software Architecture: Perspectives On An Emerging Discipline*, Prentice-Hall Publishers, New York, 1996.

37. *The Systems Engineering Process*, Course in Systems Engineering, ICTT, 2002.

38. Web site for Mathworks Matlab, Simulink modeling tools--
http://www.mathworks.com/products/

39. Web site for Project Technologies Bridgepoint tool--http://www.projtech.com/prods/index.html

40. Blanchard, B.S., and Fabrycky, W.J., *Systems Engineering and Analysis,* 1998.

41. Boehm, Barry; Port, Daniel; Basili, Victor; "Realizing the Benefits of CMMI™ with the CeBASE Method", Systems Engineering, Vol. 5, Issue 1, 2002, pp. 73-88.

42. *EIA-632: Processes for Engineering A System*, Electronics Industries Alliance, January, 1999.

43. *Systems Engineering: The Journal of the International Council on Systems Engineering*--special CMMI issue, John Wiley Publishers, vol 5, Issue 1, 2002.

44. Web site for CMU SEI CMMI--
http://www.sei.cmu.edu/cmmi/cmmi.html

45. Box, G. E. P., "Science and Statistics", J. of the American Statistical Association, 71, 1976, pp. 791-799.

46. Web site for EDS PLM SLATE tool--
http://www.eds.com/products/plm/teamcenter/slate/

47. Web site for Telelogic DOORS tool--
http://www.telelogic.com/products/doorsers/index.cfm

48. Web site for iLogix Statemate Magnum tool--
http://www.ilogix.com

49. Web site for Rational Software modeling tools--
http://www.rational.com/products/rose/index.jsp

50. Smith, Vernon; Render, Michael; Roth, Michael; Process Control Standards for Technology Development, SAE paper 981502.

51. Pinker, Steven, *The Language Instinct: How the Mind Creates Language*, William Morrow and Co., New York, NY, 1994.

52. Pattee, Howard H., ed., *Hierarchy Theory: The Challenge of Complex Systems*, George Braziller Publishers, New York, NY, 1973.

53. Web site of the International Council on Systems Engineering—http://www.incose.org

CONTACTS

William D. Schindel has been involved in system design, development, support, and evolution for over 30 years, including engineering, strategic systems marketing, and general management. He designed airborne military systems for the IBM Federal Systems Division, served on

the faculty of Rose-Hulman Institute of Technology, and has founded and run two companies focused on complex systems—one in global telecommunications and one in general embedded systems markets. Schindel holds the B.S. and M.S. degrees in Mathematics, and was awarded the honorary Doctor of Engineering degree in 2001 by Rose-Hulman Institute of Technology for his work in systems engineering. He is President of International Centers for Telecommunication Technology, Inc., Terre Haute, IN, and may be reached at schindel@ictt.com.

Vernon R. Smith is Technical Steward for System Architecture in the Electrical and Electronics Systems business unit of Caterpillar Inc. He has responsibility for defining systems engineering processes, systems architecture and communications assets for embedded control of machine and machine subsystems as well as integrated off-board systems. He holds the B.S. and M.S. degrees in electrical engineering from the University of Illinois, and has over 25 years of engineering experience at Caterpillar, including embedded electronic design, software development, and process improvement. He may be reached at smith_vern_r@cat.com.

DEFINITIONS, ACRONYMS, ABBREVIATIONS

Some systems engineering terms are harder to understand because of their everyday imprecise use or because of different meanings for the same term in other disciplines. Some of the following terms appearing in formal models are given precision not by their names (which could be replaced) or even by their prose definitions, but by their appearance in semantic models such as summarized by Figure 8. The following terms, shown alphabetically, are best explored in conceptual build-up order indicated by the sequence numbers in braces {} after each entry.

Abstraction: A description of a thing or a class of things that leaves out information of less interest in order to describe only essential aspects of interest. A distillation of essential meaning. {27}

Accounting Management: The System Management Functional Area (SMFA) concerned with managing (including reporting, analyzing, and controlling) a system based upon cost, revenue, or other financial parameters and relationships of the system's use or operation. {37}

Attribute: A characteristic of a thing, that can take on values, such as the thing's weight, color, cost, performance, priority, or status. {14}

Class: A modeled set of similar objects. {7}

Configuration Management: The System Management Functional Area (SMFA) concerned with managing the configuration of a system. The configuration of a system describes its components, their systemic relationships, and configuration related attributes (such as versions or capabilities) of the system and its components. {35}

Domain: The domain of a system is the subject matter and vocabulary used to describe interactions of the system with its environment. Domain knowledge is knowledge of behavior and environment of a system. {15}

Embedded Intelligence Pattern: The modeled system pattern that can describe intelligent behavior in all intelligent engineered or natural systems. {28}

Fault Management: The System Management Functional Area (SMFA) concerned with managing (including predicting, preventing, detecting, reporting, diagnosing, and responding to) faults of a system. Faults are defined as deviations from specified normal behavior. Fault Management is frequently concerned with minimizing system level impact of component faults. {34}

Feature: (As a Systematica Metaclass) A collection of system Functions having marketable value. In some domains, system Features are called system Services. {20}

Function: (As a Systematica Metaclass) An interaction of systems, each of which fills a functional role. {18}

Gestalt: A holistic view of a systemic pattern. {16}

Gestalt Rule™: A rule for verifying that the architectural pattern of a specific product or system meets the architectural precepts of the product line or system family to which the product or system belongs. {13}

Interface: (As a Systematica Metaclass) The association of a system, a set of functions in which it has roles, a system of access, and related views. {22}

Logical System: (As a Systematica Subclass) A system that is described by (external view of) what it does, not what it is. Equivalent to functional role. For contrast, see Physical System. {25}

MDSC: (As an Embedded Intelligence Pattern Role) A Managed System Component is a system that provides services to a System of Users, and which is managed by a Management System. See also MTSC. {30}

Metaclass: (In Systematica Metaclasses) One of the foundational or minimalist classes of things out of which all other systems engineering ideas are constructed. {8}

Metamethodology: A framework that can be configured differently to generate various specific methodologies, based on local needs, but with a common foundation of underlying ideas preserved. {5}

Methodology: A specific way of obtaining a result, as in a specific procedure or process guideline. {4}

Model: A description of a system, based upon structured information according to a modeling framework. {1}

Model based systems engineering: An approach to systems engineering that uses structured models instead of natural language prose to specify system requirements, design, or other aspects of systems. {3}

Model Interpreter: A person or automated system that interprets a Model in order to carry out some task. {2}

MTSC: (As an Embedded Intelligence Pattern Role) A Management System Component is a system that manages the performance, faults, configuration, security, or accounting aspects of a Managed System, on behalf of a System of Users. See also MDSC. {29}

Need: (As a Systematica Metaclass) An informal statement expressing a want, eligible for analysis in the specification and modeling of formal requirements. {24}

Object: A thing. Sometimes viewed as a member of a class of similar things. {9}

Pattern: A recognizable collective aspect of a system's model, emerging from the combination of its parts and their relationships. {12}

Performance Management: The System Management Functional Area (SMFA) concerned with managing (predicting, measuring, reporting, analyzing, controlling) the performance of a system. Performance is frequently associated with the central economic or functional purpose of a system, and suggests specific performance attributes, such as speed, accuracy, or productivity. {33}

Physical System: A tangible system that is described by its physical identify (what it is), not by what it does. For contrast, see Logical System. {26}

Relational: A relational perspective describes systems in terms of the relationships between their components. A relationship is any statement about components that can be true or false. Specifications of interactions between components are a common example. All relationships between things can be graphically modeled as links between graphical nodes [7]. This allows for formal expression of holistic system patterns (Gestalts). Note that relational database technology came late to this party—this is not about information technology. {10}

Security Management: The System Management Functional Area (SMFA) concerned with protecting the services of a system from unintended use, or from denial of intended use, or from corruption of intended services. {36}

Semantic Web: A web of classes and relationships between them, that can be used to establish a language, or semantic framework, used to describe things in a certain domain. {11}

SMFA: The Systems Management Functional Areas are the five specialized forms of system management. These are: Performance Management, Fault Management, Configuration Management, Security Management, and Accounting Management. See each in this list. {32}

SOAC: (As a Systematica subclass) A System of Access Component provides a means of interaction between two or more interacting systems. Examples include electronic sensors and actuators, data communication networks, human-machine interfaces, and the Internet. {21}

SOUC: (As an Embedded Intelligence Pattern Role) A System of Users Component consumes services provided by a Managed System and a Management System. {31}

SRS: A System Requirements Specification describes requirements for a system. {38}

State: (As a Systematica Metaclass) A condition of being for a system (sometimes referred to as a mode or situation) that can be conceived as having a beginning and end in time. {23}

System: (As a Systematica Metaclass) A collection of interacting components, viewed as a whole because of its systemic characteristics. {17}

Systematica™: A meta-methodology for model-based systems engineering, based on an underlying collection of Metaclasses that are used to describe both engineered systems and the processes of systems engineering, and optimized for systems engineering of families or product lines of similar but differing systems. {6}

View: (As a Systematica Metaclass) That which is exchanged between interacting systems. Most frequently consisting of information or physical entities. {19}

V1.2.25