FOUNDATIONAL CONCEPTS
FOR MODEL DRIVEN SYSTEM DESIGN

Loyd Baker, Paul Clemente, Bob Cohen,
Larry Permenter, Byron Purves, and Pete Salmon
INCOSE Model Driven System Design Interest Group

Abstract. This paper presents an initial viewpoint on an emerging technology: Model Driven System Design (MDSD). In contrast, the current state of practice has been characterized as document centered. Practitioners of system design are well aware of the technical and organizational difficulties of implementing current approaches. By modeling multiple aspects of a system throughout its life cycle, this new view of system design offers dramatic gains in productivity and product quality.

MISSION STATEMENT

To characterize model driven system design and identify transition strategies from present document driven approaches.

INTRODUCTION

Purpose. System engineers build models to better understand problems, develop candidate solutions, and validate their decisions. Different kinds of models are built to help focus on the appropriate set of questions that need answering in order to find the most reliable and cost effective solutions and to qualify the design against its requirements. The following model types are commonly used:

- **Schematic Model**: A chart or diagram, having an underlying machine-readable representation, which shows object relationships, structure, time sequencing of actions (e.g., organizational chart, spec tree, operational sequence diagram, interface diagram, state diagram, PERT network diagram, functional-flow block diagram).
- **Performance Model**: An executable structure which represents system response to external stimuli.
- **Design Model**: A machine interrogable version of the system detailed design, usually represented by CAD drawings, VHDL, C, etc.
- **Physical model**: Tangible physical equivalents used for reality experimentation and demonstration (e.g., DNA model or model airplane in a wind tunnel.)

Motivation. Customers for large systems are demanding reduced cycle time, lower development cost, increased reliability and ISO compliance. Achieving this requires a reduced error rate in design through early detection, thereby reducing the rework cost. As resources for developmental and final product testing are diminishing, application of models throughout the design process will become mandatory. With less testing and increased quality demands, an ongoing incremental approach to Validation and Verification (V&V) is critical to successful product design. This can be achieved through the combined use of Integrated Product Teams (IPT's) and by generating models of increasing fidelity that allow complete requirements compliance assessment throughout the development cycle. This overall process allows for an earlier optimization of the design and an extension of service life through retained design rationale and executable models.

Definition. A model is a limited representation of a system or process. The role of a model is to answer questions about the entity it represents. Model types may include: executable, information, design, operations, process, enterprise and organization. Models can be migrated into a cohesive unambiguous representation of a system. Verification and Validation activities interrogate the system model, then progressively iterate on adjustments to requirements and design until completeness/ quality criteria are satisfied.
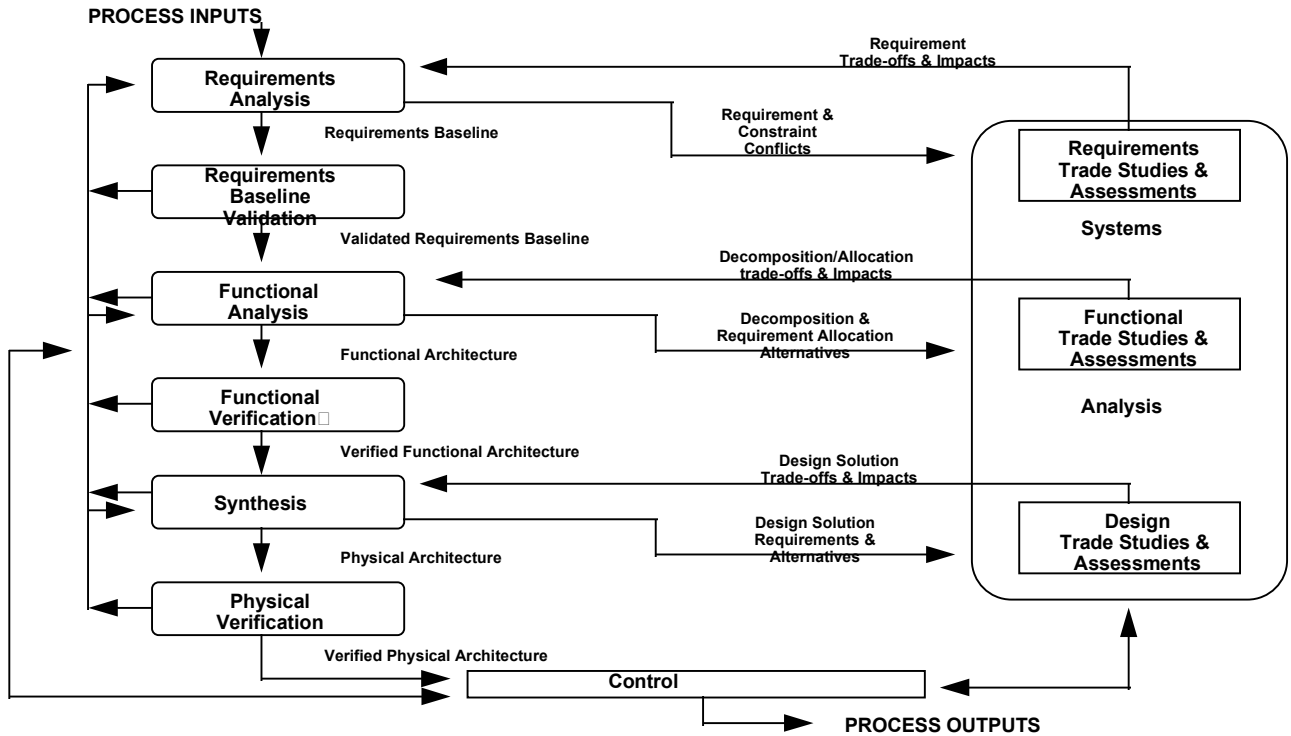
Figure 1 System Engineering Process (per IEEE 1220)

MODEL DRIVEN SYSTEM DESIGN

Process Description. The model driven approach to system design is fundamentally similar to those generally used in the industry. (See for example, Reference 1, IEEE 1220, Standard for Application and Management of the System Engineering Process). However, it prescribes some particular means of achieving desired results. Figure 1 is a summary of the system engineering process taken from Figure 5 of Reference 1.

Validation, trade studies and assessments for a Requirements Baseline, Functional Architecture, and Physical Architecture make up a large part of the identified activity. In the model driven approach these activities are to be accomplished through development of increasingly detailed models.
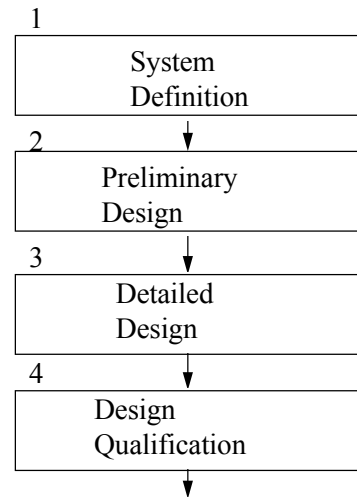

Figure 2 Development Phases of a Project

MDSD is conveniently represented as a basic subprocess which is repeated as many times as necessary. Figure 2 shows the development phases of a project. The System Definition phase corresponds to the Requirements Analysis and Requirements Baseline Validation activities of Figure 1. The Preliminary Design phase corresponds to the Functional Analysis and Functional Verification

activities. Detailed Design and Design Qualification correspond to Synthesis and part of the Physical Verification activities. Each of the main blocks in Figure 2 is further broken down as shown in Figure 3. These basic subprocesses apply to each of the development phases of Figure 2, although detailed content depends upon the actual phase. Earlier phases emphasize requirements development; later phases emphasize design drawings, source code and test.
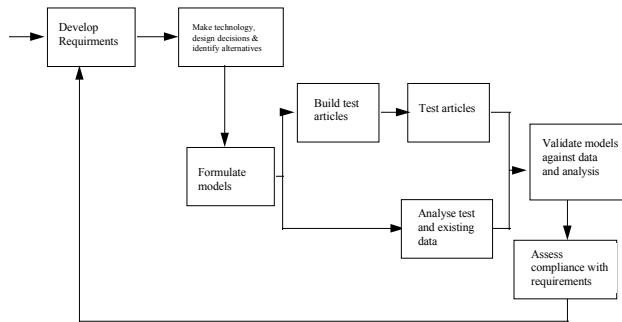
Figure 3. Subprocesses for MDSD

Some of the distinctive features of MDSD are summarized below by development phase.

System Definition. The major events of this phase are:
(a) Completion, in executable form, of system, product and subsystem interface specifications, system and product specifications, and preliminary subsystem specifications.
(b) Establishment of a machine-readable system baseline and a machine-readable preliminary subsystem "design to" baseline.
(c) Completion of a system performance model in sufficient detail to respond to all specifications in (a) above.
(d) Execution of the system model to show that the design-to baseline shows consistency of cost, schedule and technical performance requirements.
(e) Completion of technical reviews appropriate to the system definition stage, to include system model validation.

The most important new concept in this stage is that system requirements are integral to the system model, and for MDSD to work, they must be in an executable form. The role of performance oriented modeling is to assess designability of the system requirements and to make technology and architecture decisions. Testing is oriented to ensuring that models

used to make these decisions were sufficiently accurate. Customer interaction with models is used to affirm that the right system is being built.

Preliminary Design. The preliminary design phase initiates subsystem design and creates subsystem-level models, executable specifications and machine-readable design-to baselines to guide component development. Execution of the models against the design-to baseline shows preliminary compliance with specifications.

Detailed Design. The detailed design phase of the system life cycle completes subsystem design and models down to the lowest component and creates an executable component specification, model, and machine-readable build-to component baseline for each component. Execution of the models shows satisfactory preliminary compliance with performance specifications and satisfactory final compliance with design constraints.
By the completion of this stage all design decisions have been made. Except for changes, design freedoms have been exercised. The design is represented in machine-readable form, so that the detailed design (for example, a CAD model) can be interrogated for compliance with design constraints. These are limitations on the range of permitted design solutions. They include such things as dimensional limits, material selection and colors. Performance models have been validated by developmental tests and analyses, and execution of these models shows that production articles built to the detailed design will be compliant with the specifications.

Design Qualification. During this phase, performance models are validated against test data taken on test articles manufactured in accordance with the build-to baseline. Execution of the validated performance models shows satisfactory compliance with performance specifications.
Models are updated to respond to data collected during integration and test. Models are validated and approved for use in closing requirements. At functional configuration audits, requirements are checked for closure against results of model execution. By the completion of this stage, the extent of compliance of any specification requirement can be discovered by interrogating the system model.

Information View. It is also helpful to view MDSD from the kinds of information to be used and their relationships. Figure 4 shows a basic information model. The boxes represent kinds of information. Annotated lines represent relationships. Arrows show

direction of the relationship, not direction of flow of information. Bullets show a "many" relationship.
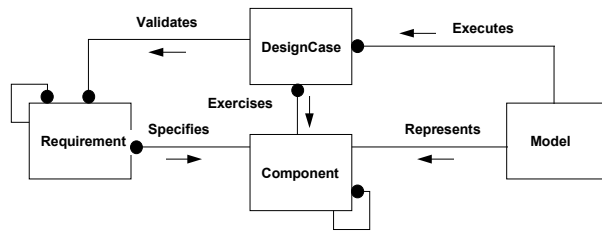

Figure 4. Information Model for Model Driven System Design

The diagram can be summarized with the following text:
- Requirements specify Components.
- Requirements may be decomposed into other Requirements
- Components may be decomposed into other Components
- Design Alternates satisfy Requirements
- Design Alternates represent Components
- Models execute Design Alternates
- Models represent Components.

In the early stages one would like to examine many Design Alternates to discover the most suitable. By the end of the Detailed Design phase, the project is considering only one Design Alternate, the build-to baseline. In early stages the models are low fidelity and geared towards decision making; eventually models become sufficiently faithful for compliance assessment.

CONTRASTING MDSD WITH DOCUMENT CENTERED SYSTEM DESIGN

Table 1 compares the anticipated benefits and challenges of MDSD to those of a document-centered approach. We base our assessments of the model driven approach on extrapolations from current experience with analytic modeling and information modeling in less integrated and extensive applications areas. The table, therefore, assumes such projections against the ideal MDSD environment. Our document centered assessments, on the other hand, are more solidly grounded in experience with current systems engineering methods and an understanding of the inherent limitations in using (a) natural language - especially unstructured text - as the primary language of discourse, and (b) document structure as the organizing paradigm.

A close examination of the benefits and challenges listed in the table reveals that modeling issues and impacts are not limited to technical characterization. Almost all of the items listed also reflect organizational, accessibility/communications, or infrastructure concerns. For example, the distribution of design information requires more than computing and networking hardware. Data representation and control strategies affect and are affected by, configuration management strategies, storage requirements, and user requirements. Similarly, usability requires a convenient paradigm for navigating a design model, which in turn may require that the modeling techniques support dynamic adaptation of the user view and tools to a user's information needs.

In a model-driven system development environment, factors that have generally been left to chance or "organizational memory" in current approaches must be addressed explicitly and in a manner that ensures consistency of design, without unduly constraining the developer's design space or behavior. As this need for explicit representation is met, our ability will improve to explore, store, retrieve, and evaluate designs easily and reliably throughout the development process. Development of a model-driven approach must therefore take into account the interdependencies of "hard" (e.g., engineering, mathematics, computability) and "soft" (e.g., organizational behavior, human-machine interaction, training) areas.

Features	Model Driven System Design	Document Centered System Design
Information Repository	Models	Documents
Reviews (SDR, PDR, CDR)	By interrogating models (automated)	Read & interpret text then compare
Verification (FCA-Functional Configuration Audit)	Implicit, incremental, automated, built into the process	Human audit process
Communication	Reproducible and consistent	Answers may depend on readers perspective
Validation	Execute in different contexts, (e.g. customer's context, on line)	Walk-throughs, reviews of paper
Traceability --- Requirements to design to verification	Integral	Accuracy is labor intensive
Reuse	Library, "Plug and Play"	Boilerplate only
Cultural Adoption	New Paradigm	Status Quo
Infrastructure:		
Workstation & Computers	Additional computing resources	Less than model driven approach
Tools	Few Available	Extensively available
Process	Immature	Processes Exist but vary from company to company
Training	Immature	Available
Navigation	Potentially easy, since relevant data is connected	Easy to browse individual documents, but not design rationale, correlation between documents is difficult

Table 1. Comparision of Model Driven and Document Driven Approaches to System Design

BENEFITS

Benefits of MDSD over textual approaches accrue from two essential features of a good model:

- **Expressiveness**. This is the power to express complex information in ways that are easily understood. Models can achieve this expressive power through physical representations, graphics, animation, 3-D representations, and the use of color.

- **Rigor**. Compared with textual representations, executable models provide clear and unambiguous definitions of behavior, capability or design. This is a consequence of the usual practice of building hierarchical models from primitives that are both rigorous and unambiguous.

These benefits require an investment by the program community into understanding the language of the model. This investment is not currently required on text-based programs because English language skills are normally available in all employees. The MDSD approach on a program will typically require one or two highly trained process and tool experts who are well versed in the methodology. These experts will then train and mentor the program staff. Typically, a program engineer can be productive with modeling approaches after one half day of training by a process expert..

It is useful to consider the benefits separately from the perspectives of Customer and Supplier.

Customer Benefits. Customers benefit from better overall cost, schedule and technical performance on programs. This primarily from improved customer/supplier communications. Improved communications have the following forms:

- More effective translation of user needs into program requirements via the expressiveness and rigor of models. This means that customers are more likely to get what is needed, as opposed to what is specified in textual documents, which may be both voluminous and flawed.

- Improved visibility into program performance because these results can be available continuously throughout the program; they are not just snapshots which are studied in formal program reviews. There is also the potential for results of executable models to be more intuitively understandable

- Early problem discovery leads to collaborative solutions between customer and supplier. These solutions can be incorporated much more effectively as the program proceeds, rather than during the crisis atmosphere of final system sell-off.
- Issues and trades are visible to support decision making.
- Greater supplier accountability results from inherent progress visibility.
- Availability of validated models for qualified components encourages reuse.

Supplier Benefits. Because of the expressiveness of the models, intra-program communications can improve dramatically. Using text-based processes with IPTs can result in a great amount of time spent reaching a common understanding of the design between and among the different disciplines comprising the IPT. If models are jointly developed in a concurrent engineering environment and shared across an electronic network, this communications demand on design engineers can be greatly reduced. For the greatest benefit, several modern concepts may be integrated with the modeling process. These include concurrent engineering, object oriented design, and on-line communications between program engineers.

Supplier benefits can be enumerated as follows:
- Hierarchical decomposition of models supports visibility of information at its level of relevance. The associated "de-cluttering" of design information is extremely effective in enabling engineers to "see" the critical issues at a particular design level.
- More exhaustive search for optimal solutions is possible.
- Rigor of the models helps to avoid ambiguities, mistakes, and rework.
- Status of designs, processes and compliance is visible and traceable as a direct result of the model.
- Models provide linkage between hardware, software, and other design elements. This is important throughout the life cycle. It enables system level interfacing errors to be identified early and avoids surprises during the Design Qualification phase.
- Reuse benefits are similar to those for the customer.

Transition Considerations . Based on experience gained from implementing system engineering automation tools it is necessary to plan for their introduction and use. Successful transition to MDSD will require the following considerations:
- Adequate resources must be applied to infrastructure and training.
- A well written system design process is necessary for effective communication. It should include:
  - Description of interfaces between technical disciplines
  - Description of task completion criteria.

CONCLUSIONS AND RECOMMENDATIONS

Model driven system design offers a new rigor in specifying and verifying systems. It supports continuous assessment of consistency between requirements and design. Customers are more likely to receive the system they really need because of the opportunity to experiment with requirements and design options.

Model driven system design is a new way of doing business. It requires a different mind set and a different skill set from traditional approaches. This new method is however, not yet fully defined. In fact its definition is only just beginning. Readers of this paper are invited to respond and participate in developing this epoch making change to system engineering.

INCOSE is in a unique position to set the industry standard for this emerging approach to system design. INCOSE should provide guidance for infrastructure preparation, training, and standard interfaces to requirements, models and designs.

REFERENCE

IEEE 1220, Standard for Application and Management of the Systems Engineering Process, 1995